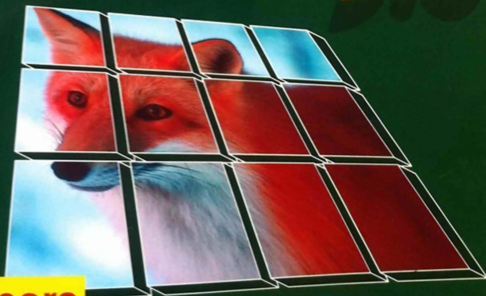
Bazele Visual FoxPro 5.0



Gabriel Dima
Mihai Dima



Teora

Bazele Visual FoxPro 5.0

Lucrarea tratează Sistemul de Gestiune a Bazelor de Date (SGBD) Visual FoxPro, versiunea 5.0. Sunt descrise atât limbajul pe baza căruia a fost construit Visual FoxPro, cât și elementele de interfață ale mediului. Datorită diferențelor minore între versiunile Visual FoxPro 3.0 și Visual FoxPro 5.0, o mare parte a celor prezentate în lucrare sunt utile și celor care folosesc versiunea 3.0.

Cartea nu este un ghid complet pentru utilizatorii de Visual FoxPro, deoarece o asemenea lucrare ar fi mult mai voluminoasă. Sunt tratate acele componente care stau la baza SGBD, iar subiectele abordate sunt suficiente pentru construirea unui sistem informatic de complexitate medie.

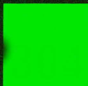
Accentul se pune pe orientarea pe obiecte a limbajului FoxPro, deoarece aceasta reprezintă o tendință generală a limbajelor de programare moderne. De asemenea, sunt prezentate în detaliu uneltele interactive folosite la construirea diferitelor elemente – este vorba despre un limbaj „vizual".

Sunt abordate și tehnici speciale disponibile în Visual FoxPro, caracteristice sistemelor de operare moderne Windows 95 și Windows NT:

- Tehnologia OLE
- Schimbul dinamic de date între aplicații
- Aplicațiile client/server

Pentru: ■ începători
■ cei care cunosc versiunile anterioare de FoxPro

Editura **Teora**
CAL VISUAL FOXPRO 5.0, AR
Cod: 0298
Lei 60.000




ISBN 973-20-0298-0
9 789732 002988

Cuprins

Partea I – INTRODUCERE

Introducere

Capitolul 1

- Prezentarea generală a cărţii
- Cum să citiţi această carte?
- Programarea structurată, programarea orientată spre obiecte, programarea condusă de evenimente şi programarea vizuală
- Construirea şi executarea programelor

Prezentarea generală a cărții

Cartea de față tratează Sistemul de Gestiune a Bazelor de Date Visual FoxPro, versiunea 5.0. Sunt descrise atât limbajul pe baza căruia a fost construit SGBD-ul, cât şi elementele de interfață ale mediului. Datorită diferențelor minore între versiunile Visual FoxPro 3.0 şi Visual FoxPro 5.0, considerăm că o mare parte a celor prezentate în lucrare sunt utile şi celor care folosesc versiunea 3.0 a sistemului.

Cartea nu este un ghid complet pentru utilizatorii Visual FoxPro, deoarece o asemenea lucrare ar ocupa un spațiu mult mai mare decât cel al lucrării de față, ci tratează acele componente care stau la baza sistemului. Dar subiectele abordate sunt suficiente pentru construirea unui sistem informatic de complexitate medie, urmând ca profesioniştii să apeleze la alte lucrări mai specializate şi mai detaliate.

În carte se pune accentul pe orientarea spre obiecte a limbajului FoxPro, deoarece aceasta reprezintă o tendință generală a limbajelor de programare moderne. De asemenea, sunt prezentate preponderent uneltele interactive folosite la construirea diferitelor elemente (doar este vorba de un limbaj „vizual").

Sunt abordate şi câteva dintre tehnicile speciale disponibile în Visual FoxPro, caracteristice sistemelor de operare moderne, precum Windows 95 şi Windows NT. Acesta este, de exemplu, cazul comunicării dinamice între aplicații, al modelului client/server de construire a aplicațiilor etc.

Cum trebuie citită această carte?

Vom împărți cititorii acestei cărți în două categorii: cei care au folosit anterior una dintre versiunile 2.6 pentru DOS sau Windows ale sistemului de gestiune a bazelor de date FoxPro şi cei care pornesc de la zero, adică fac cunoştință cu lumea bazelor de date prin intermediul programului Visual FoxPro (3.0 sau 5.0).

Pentru cei care cunosc FoxPro 2.6

Pentru prima dintre categorii, adică pentru aceia care au mai lucrat în FoxPro 2.6 pentru DOS sau pentru Windows, este necesară o adaptare, pe de-o parte la noul stil de programare (orientat spre obiecte, condus de evenimente şi vizual) şi, pe de alta, la noile unelte disponibile în mediu. Prin urmare, recomandăm citirea capitolelor de interes în ordinea dorită de cititor, în continuare fiind prezentate pe scurt câteva dintre schimbările la care trebuie să se adapteze un astfel de utilizator.

Prima schimbare majoră pentru un utilizator al sistemului FoxPro 2.6 în trecerea sa la Visual FoxPro este folosirea modelului programării orientate spre obiecte în locul celui al programării structurate. Cu toate că, din motive de compatibilitate, Visual FoxPro

permite şi folosirea modelului programării structurate, recomandăm cu căldură trecerea la noul model, mai avantajos, mai modern şi mai de viitor.

O altă schimbare majoră este trecerea la elemente de programare vizuală. Deşi în versiunile 2.6 pentru DOS şi Windows existau unelte de proiectare interactivă (Generatoarele), cele din Visual FoxPro sunt adaptate modelului orientării spre obiecte şi modelului programării conduse de evenimente. Aceste unelte sunt mult modificate faţă de rudele lor mai vechi şi deci ele trebuie studiate în amănunt (de exemplu, Constructorul de forme, care este cu totul diferit de Generatorul de ecrane din FoxPro 2.6).

Alte schimbări apar în conceperea bazelor de date. Pe lângă noile tipuri de date, care permit adaptarea mai fină la situaţiile practice, a fost introdus conceptul de bază de date container, care, pe lângă tabele şi relaţiile între acestea, conţine o mulţime de alte elemente, precum vederi, conexiuni etc. De asemenea, la tabele şi la bazele de date pot fi ataşate acum secvenţe de cod care să fie executate la apariţia diferitelor evenimente, ceea ce face ca o dată cu proiectarea bazelor de date să se proiecteze şi modul de prelucrare a acestora.

O altă adaptare necesară în trecerea la Visual FoxPro este înlocuirea, pe cât posibil, a prelucrărilor orientate spre înregistrări cu prelucrări orientate spre grupuri de înregistrări. Se vor folosi astfel interogările şi deci limbajul SQL în locul vechilor metode de manipulare a datelor din bazele de date (GOTO, REPLACE, DELETE...). Această schimbare vine şi în sprijinul proiectării aplicaţiilor client/server, un model foarte performant şi de actualitate în domeniul bazelor de date.

Pentru cei noi în lumea bazelor de date şi a SGBD-urilor

Pentru acea categorie de cititori care nu au mai avut anterior un contact cu FoxPro sau cu alte SGBD-uri relaţionale, recomandăm citirea capitolelor în ordinea în care sunt ele prezentate în carte. Desigur că se pot sări o serie de detalii (prezentarea unor funcţii, unelte etc.), care urmează a fi descifrate atunci când apar situaţiile practice respective. După parcurgerea tuturor capitolelor cărţii, cititorul îşi va putea construi singur propriul sistem informatic.

Programarea structurată, programarea orientată spre obiecte, programarea condusă de evenimente şi programarea vizuală

De-a lungul timpului, tehnicile şi metodele de programare a calculatoarelor au evoluat, începând cu programarea clasică, continuând cu cea structurată, trecând apoi la programarea orientată spre obiecte şi la cea condusă de evenimente şi ajungând astăzi la modelul programării vizuale, implementat în majoritatea mediilor moderne de programare.

Diferenţele dintre aceste modele nu se află în rezultatele furnizate de programe, ci în modul de concepere a programelor respective. Cu alte cuvinte, aceeaşi operaţie care se execută pe un sistem de calcul se poate realiza prin oricare dintre modelele amintite. Diferă în schimb efortul necesar realizării programului care să rezolve problema, cunoştinţele necesare, timpul consumat, viteza de execuţie etc.

În esenţă, un calculator funcţionează pe baza unui program construit după un model clasic, liniar. Procesorul calculatorului înţelege o gamă limitată de instrucţiuni (microinstrucţiuni), care sunt executate în ordinea găsirii lor în memorie. Există însă o grupă de instrucţiuni care permit salturi la alte instrucţiuni decât la cea care urmează, salturi condiţionate eventual de îndeplinirea unei anumite condiţii. Acestea sunt aşa numitele instrucţiuni `GOTO` (mergi la), pe baza lor fiind construite primele modele de programare.

De exemplu, limbajul BASIC clasic conţinea instrucţiunea `GOTO`, care permitea realizarea unui salt la o anumită linie. În combinaţie cu instrucţiunea `IF` se realizau salturi condiţionate, astfel că se obţinea tot ce era nevoie pentru construirea unui program.

Dezavantajul major al acestui model a fost dificultatea depanării programelor care depăşeau o anumită dimensiune. Deseori, la depanare apăreau situaţii când, după urmărirea atentă a unei secvenţe de instrucţiuni, se sărea brusc în altă zonă a programului, făcând urmărirea foarte dificilă.

Următoarea schimbare majoră a fost dată de apariţia **modelului programării structurate**, prin care se eliminau complet salturile necontrolate. În locul acestora au fost introduse structuri speciale numite bucle, care ofereau o lizibilitate mult sporită.

Exemplu

De exemplu, o buclă de tipul:

```
Execută
  <instrucţiuni>
până când este îndeplinită <condiţie>
```

poate fi construită şi cu ajutorul modelului clasic de programare, ca mai jos:

```
* <instrucţiuni>
Dacă nu este îndeplinită <condiţie> salt la *
```

Conform modelului programării structurate, orice sarcină (orice program) poate fi îndeplinită cu ajutorul a trei structuri standard: cea liniară (execuţia instrucţiunilor una după alta), cea condiţională (IF – dacă) şi cea iterativă, bucla, fie cu test final (DO... UNTIL... – execută... până când...), fie cu test iniţial (WHILE... DO... – cât timp... execută...).

Toate cele trei structuri pot fi construite cu ajutorul modelului clasic (cu GOTO şi IF) şi, prin urmare, compilarea programelor structurate nu a reprezentat o dificultate pentru producători. Probleme au fost mai mult pentru programatori, care au fost nevoiţi să se adapteze la noul stil de programare şi la noile reguli (mai stricte) impuse de acest model.

Modelul programării structurate a reprezentat un pas important înainte, nu datorită unei creşteri de viteză sau unor facilităţi suplimentare, ci datorită creşterii clarităţii şi lizibilităţii programelor, ceea ce a dat posibilitatea construirii unor programe din ce în ce mai mari, de sute şi chiar mii de linii. Modelul programării structurate este implementat şi în FoxPro 2.6 şi chiar şi în Visual FoxPro, dar această din urmă versiune are implementat şi modelul programării orientate spre obiecte.

Un nou pas major în programare a fost introducerea **modelului programării orientate spre obiecte**. Conform acestui model, un program reprezintă o serie de definiţii de date şi de algoritmi de prelucrare a acestora, asamblaţi împreună în entităţi numite obiecte. În locul unei succesiuni de algoritmi care prelucrează date externe s-au introdus entităţi care includeau atât datele, cât şi procedurile cu care acestea erau prelucrate. Însă modelul programării structurate nu a dispărut, în cadrul metodelor fiind folosite în continuare structurile sale de programare.

Din nou, nu s-a realizat ceva care nu putea fi făcut prin vechiul model de programare, orice program construit după modelul programării orientate spre obiecte putând fi construit şi prin modelul programării structurate. Avantajele programării orientate spre obiecte sunt însă esenţiale pentru lucrul în echipă, atunci când reutilizarea codului este esenţială, în cazul programelor de dimensiuni mari etc.

Începând cu versiunea Visual FoxPro 3.0 a fost implementat modelul programării orientate spre obiecte, aducând astfel SGBD-ul FoxPro şi limbajul aferent la nivelul celor mai moderne medii de programare

Alături de modelul programării orientate spre obiecte a fost dezvoltat şi **modelul programării conduse de evenimente**. Conform acestui model un program reprezintă un ansamblu de proceduri care nu sunt apelate după o anumită regulă temporală, ci sunt lansate în execuţie numai atunci când în sistem apar anumite evenimente. Prin urmare, dacă în modelul clasic programarea era de tipul:

Vezi dacă... şi, dacă da, execută...

în modelul programării conduse de evenimente un program este o succesiune de secvenţe de tipul:

Dacă apare evenimentul... execută...

Apariţia evenimentelor este arbitrară pentru programul respectiv, el fiind doar învăţat cum să răspundă atunci când survin acestea.

Elemente de programare condusă de evenimente au fost implementate în FoxPro încă de la versiunile pentru DOS, prin comenzi de tipul ON LABEL, care permiteau utilizatorilor să specifice modul în care programele lor să răspundă la diferite combinaţii de taste, fără a şti exact când vor apărea aceste evenimente. O dată cu apariţia versiunilor Visual FoxPro 3.0 şi apoi 5.0, modelul programării conduse de evenimente a fost mai bine implementat, fiind extinsă gama evenimentelor la care poate răspunde programul utilizatorului.

Limbajul de programare implementat în Visual FoxPro este unul orientat spre obiecte, dar totodată condus de evenimente (cele două caracteristici neexcluzându-se reciproc, ba mai mult chiar, convieţuind foarte bine împreună).

O dată cu avântul luat de mediile de dezvoltare de programe au apărut unelte din ce în ce mai complexe, care să ajute la scrierea mai rapidă a programelor, să preia sarcinile de rutină ale programatorilor şi să contribuie la organizarea muncii acestora. Unele dintre aceste unelte permit programatorilor să specifice în mod interactiv diferite opţiuni şi să construiască tot în mod interactiv diferite elemente, urmând ca pe baza acestora să fie generate programe.

Aceste unelte care permit scrierea programelor într-un mod cvasi-interactiv au condus la un nou stil de programare numit **programare vizuală**. Prin urmare, programarea vizuală constă, de fapt, în folosirea unor utilitare care permit programatorilor să înlocuiască, acolo unde este posibil, scrierea directă a codului cu specificarea interactivă a opţiunilor corespunzătoare.

Programarea vizuală este construită pe baza unui model de programare – cel orientat spre obiecte – fără a-l înlocui pe acesta. Metoda vizuală doar ajută la construirea mai rapidă a diferitelor elemente ale unei aplicaţii, folosind diferite unelte interactive, având însă în spate un model de programare.

Cu toate acestea, încă nu s-a ajuns la stadiul în care programarea prin cod să dispară în totalitate, uneltele prezente în mediile moderne de dezvoltare reprezentând combinaţii între cele două metode, interactivă şi prin codificare directă.

Elemente de programare vizuală se găsesc în FoxPro încă de la versiunile pentru DOS. Acestea sunt, de exemplu, Generatorul sau Constructorul de ecrane ori cel de rapoarte, care există încă de la versiunea FoxPro 2.0.

În Visual FoxPro există unelte vizuale pentru construirea majorităţii elementelor unei aplicaţii, forme, meniuri, rapoarte, baze de date, interogări etc. De asemenea,

Visual FoxPro conţine o serie de „Vrăjitori", care permit construirea unor tipuri standard de elemente cu un minim de efort din partea utilizatorului (acesta din urmă trebuie doar să răspundă la câteva întrebări ale sistemului).

În concluzie, Visual FoxPro reprezintă un mediu de dezvoltare modern, bazat pe modelul programării orientate spre obiecte şi pe cel al programării conduse de evenimente, la care se adaugă uneltele specifice programării vizuale.

Construirea şi executarea programelor

Vom prezenta aici modul în care poate fi construit un program, pentru a putea astfel testa exemplele care apar în această carte.

Def

*Un **program** reprezintă o succesiune de instrucţiuni, realizată în conformitate cu regulile limbajului de programare folosit, care rezolvă o anumită problemă, îndeplineşte o anumită sarcină, printr-un anumit algoritm.*

Scrierea unui anumit program înseamnă stabilirea instrucţiunilor care alcătuiesc programul, a ordinii acestora în cadrul programului şi transmiterea lor spre calculator.

Un program este depus pe disc într-un fişier. Crearea unui fişier pentru un nou program se realizează printr-o comandă de tipul:

```
MODIFY COMMAND <nume program>
```

Extensia implicită a fişierului nou creat este `.PRG` (atribuită automat de sistem, dacă nu se specifică altfel). Ca urmare a acestei comenzi, pe ecran este deschisă o fereastră de editare, în care utilizatorul poate introduce conţinutul fişierului respectiv. Ieşirea din editare, cu salvare, se realizează prin apăsarea combinaţiei de taste Ctrl+W, iar fără salvare, prin tasta Esc.

Când se doreşte execuţia instrucţiunilor programului, fişierul este convertit într-o formă intermediară (operaţie care poartă numele de **compilare**), care este apoi interpretată de sistem. Execuţia unui program se realizează prin comanda `DO`:

```
DO <nume program>
  WITH <listă parametri>
```

Obs

*Înainte de execuţia unui program, acesta este compilat. Sistemul preia **programul sursă** (`.PRG`) şi îl converteşte într-o **formă compilată**, care va fi interpretată de FoxPro. Această formă compilată reprezintă tot un fişier pe disc, cu acelaşi nume cu al programului sursă, dar cu extensia `.FXP`.*

Modul de execuţie a unui program poate depinde de parametrii externi, transmişi programului la lansarea în execuţie. De exemplu, un program de sortare a unui fişier

necesită ca parametri externi numele fişierului de sortat şi ordinea sortării (crescătoare, descrescătoare). La execuţia unui program prin comanda DO, parametrii de rulare sunt transmişi acestuia prin <lista parametri> a clauzei WITH.

Execuţia unui program se va opri în una din următoarele situaţii:

- la execuţia unei comenzi RETURN, CANCEL, QUIT;
- când se întâlneşte sfârşitul fişierului;
- când se întâlneşte o altă comandă DO (după executarea noului program, cu această comandă se revine în programul apelant).

Comanda RETURN termină execuţia programului returnând controlul la programul apelant. Dacă RETURN este urmată de o expresie, atunci valoarea rezultată din evaluare se returnează programului apelant. Comanda CANCEL determină sfârşitul execuţiei programului curent şi predarea controlului în fereastra de comenzi.

Suspendarea execuţiei unui program (cu posibilitatea de reluare) se realizează prin comanda SUSPEND. Continuarea rulării unui program suspendat se face prin comanda RESUME.

Execuţia unui program se încheie şi la întâlnirea comenzii QUIT, care determină ieşirea din mediul Visual FoxPro.

Tipuri de date gestionate în Visual FoxPro

Capitolul 2

- Generalități
- Tipul logic (boolean)
- Tipuri de date numerice
  - ✓ Prezentare generală
  - ✓ Prelucrarea datelor numerice
- Tipul șir de caractere
  - ✓ Prezentare generală
  - ✓ Prelucrarea șirurilor de caractere
- Tipuri de date pentru gestiunea timpului
  - ✓ Tipul dată calendaristică
  - ✓ Tipul moment de timp
- Tipuri de date speciale
  - ✓ Tipul „memo"
  - ✓ Tipul „general"
- Conversii între tipurile de date
- Funcții de uz general

Generalităţi

Datele reprezintă informaţii care fac obiectul prelucrărilor automate în sistemele de calcul. Fiecare dată este memorată pe suporturile de memorare ale sistemelor de calcul într-un anumit format. Pe de altă parte, interpretarea valorilor dintr-o zonă de memorie se face diferit, în funcţie de semnificaţia datelor respective.

Exemplu

Să presupunem că într-o zonă de memorie de 8 octeţi este depozitată o dată calendaristică (30 aprilie 1999). Următoarea figură ilustrează modul în care este memorată data respectivă.

Număr octet	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Valoare memorată	*1*	*9*	*9*	*9*	*0*	*4*	*3*	*0*
	anul				*luna*		*ziua*	

Cunoscând că în zona respectivă se află memorată o dată calendaristică, adunarea lui 1 la valoarea existentă va duce la următoarea situaţie:

Număr octet	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Valoare memorată	*1*	*9*	*9*	*9*	*0*	*5*	*0*	*1*
	anul				*luna*		*ziua*	

în care se observă că nu s-a adăugat 1 la numărul zilei, ci s-a trecut la ziua următoare în ordine calendaristică (1 mai 1999). Dacă însă valoarea memorată în zona respectivă ar fi reprezentat o sumă de bani, adunarea ar fi condus la situaţia prezentată mai jos:

Număr octet	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Valoare memorată	*1*	*9*	*9*	*9*	*0*	*4*	*3*	*1*

Prin urmare, este important atât modul de memorare a datelor (formatul fizic de reprezentare), cât şi semnificaţia lor. Aceste caracteristici ale unei date sunt precizate prin tipul său.

Def ***Tipul unei date*** *este o caracteristică ce stabileşte modul în care data este înregistrată pe suportul de memorare şi modul în care este interpretată şi prelucrată.*

În orice limbaj există o serie de tipuri de date predefinite şi, de obicei, se prevede posibilitatea definirii de noi tipuri de date pe baza celor existente deja. În Visual FoxPro sunt implementate următoarele tipuri de date:

- tipul logic (boolean)
- tipuri de date numerice
  - tipul numeric simplu
  - tipul numeric dublu
  - tipul întreg
  - tipul monetar
- tipul şir de caractere
- tipuri de date pentru gestiunea timpului
  - tipul dată calendaristică
  - tipul moment de timp
- tipul „memo"
- tipul „general"

În cele ce urmează vor fi prezentate pe rând aceste tipuri de date, împreună cu cele mai importante funcţii folosite pentru prelucrarea datelor de tipurile respective.

Tipul logic (boolean)

Tipul logic sau boolean este folosit în Visual FoxPro pentru gestiunea datelor ce pot lua doar două valori: *adevărat* (în engleză „true") sau *fals* (în engleză „false"), *da* sau *nu*, *pozitiv* sau *negativ* etc. Pentru a specifica valoarea *adevărat* a unei expresii de tip logic se foloseşte construcţia **.T.** (de la **True**), iar pentru valoarea *fals* se foloseşte **.F.** (de la **False**).

O expresie de tip logic reprezintă o combinaţie de operanzi şi operatori, construită după anumite reguli sintactice, a cărei evaluare va avea ca rezultat o valoare logică.

Operanzii ce intră în componența expresiilor logice pot fi câmpuri (de tip logic) ale unei tabele, funcții ce returnează valori logice, variabile de tip logic şi constantele prezentate mai sus.

Operatorii logici, în ordinea priorității de evaluare, sunt sintetizați în următorul tabel:

Operator	Semnificație
(,)	grupează expresiile
!, NOT	negație logică
AND	şi logic
OR	sau logic

Rezultatul unei expresii care conține operatori relaționali (<, >=, ..., vezi paragrafele următoare) este tot de tip logic.

Exemplu

```
? NOT (1=3)
.T.
? (1<=4) AND (5>3)
.T.
? 6<3 OR 4*2=9
.F.
```

Tipuri de date numerice

Prezentare generală

Tipurile de date numerice sunt folosite, evident, pentru manipularea numerelor. În FoxPro, versiunea 2.6, exista un singur tip de date numeric. În Visual FoxPro însă au fost introduse tipuri de date numerice suplimentare, pentru a asigura o mai mare elasticitate în stocarea şi manipularea acestor date.

Pentru datele numerice stocate în memoria internă a sistemului (deci în variabile) se foloseşte însă un singur mecanism de memorare şi prelucrare. Astfel, orice variabilă numerică ocupă în memoria calculatorului 8 octeți, iar calculele numerice se realizează cu o precizie de aproximativ 16 cifre zecimale.

Obs

Excepţie de la această regulă face tipul monetar, care în memorie este reprezentat tot pe 8 octeţi, a căror interpretare este însă alta decât la celelalte tipuri de date numerice. Mai mult, după cum vom vedea, calculele cu tipul monetar se fac cu o precizie de 4 cifre zecimale.

În cazul câmpurilor numerice ale tabelelor se pot folosi patru tipuri de date: numeric simplu, numeric dublu, întreg şi monetar, caracteristicile fiecăruia dintre ele fiind prezentate pe scurt în cele ce urmează.

Tipul numeric simplu („numeric" sau „float")

Acesta este tipul numeric clasic, implementat şi în versiunile FoxPro anterioare. În tabele, zona de memorie alocată pentru un câmp de acest tip este dependentă de lungimea declarată de utilizator la crearea tabelelor respective (de la 1 la 20 de octeţi). Pentru fiecare cifră sistemul alocă în tabelă un octet, în care este memorat codul ASCII al cifrei zecimale respective. Acest mod de memorare este ineficient, fiind folosit mai ales pentru compatibilitatea cu tabelele construite în versiunile anterioare ale SGBD-ului, sau pentru valori raţionale mici (şi cu precizie mică).

Exemplu

De exemplu, un câmp de acest tip, cu lungimea de 2, poate memora valori între 0 şi 99, având deci în total 100 de variante. Dacă însă s-ar folosi întreaga capacitate fizică a unei zone de memorie de 2 octeţi (cum este cazul altor tipuri de date numerice), s-ar putea memora numere de la 0 la 65535, deci în total 65536 variante. Diferenţa este semnificativă.

Tipul numeric dublu („double")

Datele de acest tip reprezintă numere memorate în virgulă mobilă, cu dublă precizie. Lungimea zonei de memorie ocupate de aceste date în tabele este fixă, de 8 octeţi. Acest tip de date se foloseşte pentru memorarea numerelor foarte mari (până la ordinul 10^{300}) sau a celor pentru care se doreşte o precizie foarte mare (până la 13 zecimale).

Deşi acest tip de date asigură o bună utilizare a memoriei, există situaţii în care nu este recomandat, datorită dimensiunii fixe impuse. Să luăm, de exemplu, o tabelă în care este necesară memorarea greutăţii unor persoane. Dacă pentru câmpul respectiv s-ar stabili tipul numeric dublu, atunci fiecare dată va ocupa în tabelă opt octeţi. Dacă însă pentru acest câmp s-ar stabili tipul numeric simplu, cu lungimea totală de 6 octeţi (3 cifre înaintea punctului zecimal şi 2 cifre după punct), s-ar obţine o economie de 2 octeţi pe înregistrare.

Tipul întreg („integer")

Acest tip de date este specific numerelor întregi, cuprinse în intervalul [–2147483647, +2147483646]. Datele de tip întreg ocupă 4 octeţi, această dimensiune fiind (ca şi în cazul anterior) fixă.

Ori de câte ori avem de memorat într-o tabelă valori întregi mai mari de 9999, este indicat a se folosi acest tip de date. Dacă însă valorile sunt mici, se poate folosi tipul numeric simplu. De exemplu, pentru memorarea vârstei unei persoane (care ia valori în intervalul [0, 150]), este de preferat folosirea tipului numeric simplu, cu lungimea de 3, fără zecimale (se economiseşte un octet pe înregistrare).

Dacă însă trebuie să memorăm salariile angajaţilor unei unităţi economice, mai indicat este tipul de date întreg, deoarece salariul poate lua valori în intervalul [100000, 10000000]. Tipul numeric simplu impune alocarea a 8 octeţi pentru fiecare înregistrare, pe când, pentru tipul întreg, evident că zona respectivă este de doar 4 octeţi (o economie de 4 octeţi pe înregistrare).

Tipul monetar („currency")

Tipul monetar este folosit pentru memorarea valorilor exprimate în bani. Zona de memorie alocată are dimensiunea de 8 octeţi. O valoare de acest tip se declară prin introducerea în faţa valorii numerice respective a simbolului monetar ($). De exemplu, următoarea instrucţiune determină definirea variabilei a de tip monetar şi atribuirea valorii de 500,47.

```
a=$500.47
```

Precizia folosită pentru memorarea acestor date este de maximum 4 zecimale. Valorile de acest tip trebuie să se încadreze între aproximativ $-9 \cdot 10^{14}$ şi $+9 \cdot 10^{14}$.

Prelucrarea datelor numerice

La prelucrarea datelor numerice este deosebit de important modul de evaluare a expresiilor în care intervin operanzi numerici (constante, variabile, câmpuri ale unor tabele sau funcţii) sau a acelora al căror rezultat este de tip numeric.

Operatorii care se aplică unor operanzi numerici, având ca rezultate tot valori numerice, sunt sintetizaţi în tabelul următor, în ordinea priorităţilor de evaluare:

Operator	Semnificație
(,)	grupează expresiile
**, ^	ridicare la putere
*, /	înmulțire, împărțire
%	modulo (restul împărțirii)
+, -	adunare, scădere

În acest tabel prioritatea scade de sus în jos, pentru operațiile cu același nivel de prioritate evaluarea făcându-se de la stânga la dreapta, în ordinea apariției operatorilor în expresie.

Exemplu

Astfel, o expresie de tipul:

```
(2*3)^2-4+7%3*2
```

se evaluează după cum urmează:

```
6^2-4+7%3*2
36-4+7%3*2
36-4+1*2
36-4+2
32+2
34
```

Între două expresii numerice se pot aplica, de asemenea, operatori relaționali, obținându-se astfel expresii logice. Acești operatori sunt prezentați în următorul tabel:

Operator	Semnificație
<	mai mic decât
>	mai mare decât
=	egal cu
<>, #, !=	diferit de
<=	mai mic sau egal cu
>=	mai mare sau egal cu

Să vedem acum câteva funcţii care au ca obiect de prelucrare date numerice sau care returnează astfel de valori.

Funcţia `MOD()` este echivalentă cu operatorul `%`, ea returnând restul obţinut prin împărţirea primei expresii numerice primite ca argument la cea de-a doua.

Exemplu

```
? MOD (38,6)
2
? MOD (44.44,11.11)
0
```

Funcţii referitoare la semnul datelor numerice

Pentru aflarea valorii absolute a unui număr se foloseşte funcţia `ABS()`, iar semnul unei valori numerice este întors de funcţia `SIGN()`. Aceasta din urmă returnează:

- +1 dacă expresia numerică trimisă ca argument este pozitivă;
- 0 dacă expresia numerică respectivă este nulă;
- –1 dacă expresia numerică este negativă.

Exemplu

```
? ABS (-400)
400
? SIGN (-32)
-1
a=-2/3
? a=SIGN(a)*ABS(a)
.T.
```

Funcţii de aproximare a datelor numerice

Partea întreagă a unei expresii numerice este returnată de funcţia `INT()`, iar partea fracţionară a acesteia se obţine cu ajutorul aceleiaşi funcţii, scăzându-se din număr partea sa întreagă.

Exemplu

```
? INT (14.46)
14
? INT (-2.25)
-2
```

```
a=14.46
? a-INT(a)
0.46
a=-2.25
? a-INT(a)
-0.25
```

Funcţia `ROUND()` realizează, de asemenea, o aproximare a unui număr, dar nu neapărat la un întreg, ca funcţia anterioară, ci la un număr raţional cu un număr dat de zecimale. Funcţia primeşte ca parametri două expresii numerice, prima reprezentând expresia de rotunjit, iar cea de-a doua numărul de zecimale ce se vor păstra în valoarea returnată de funcţie.

Funcţiile matematice elementare

Din categoria acestor funcţii fac parte: exponenţiala (e^x), logaritmul natural (ln x), logaritmul zecimal (log x) şi radicalul ($\sqrt{x}$). Funcţiile FoxPro corespunzătoare sunt `EXP()`, `LOG()`, `LOG10()` şi `SQRT()`.

Exemplu

```
? EXP (2)
7.39
? LOG (2)
0.69
? LOG10 (10)
1.00
? EXP(LOG(3))
3.00
? SQRT (2)
1.41
```

Cu ajutorul acestor funcţii se pot executa majoritatea calculelor matematice.

Exemplu

$log_b a$ se obţine prin formula $log_b a = ln\ a / ln\ b = log\ a / log\ b$

y^x se obţine prin $y^x = e^{x \ln y}$ sau y^x

Funcţii trigonometrice

FoxPro are implementate atât funcţiile trigonometrice directe, cum sunt sinus, cosinus şi tangentă, cât şi funcţiile inverse, arcsin, arccos şi arctangentă. Funcţiile trigonometrice operează cu unghiuri, care pot fi exprimate atât în grade (sexagesimale), cât şi în radiani. Conversia între cele două unităţi de măsură se realizează cu ajutorul funcţiilor `DTOR()` (din grade în radiani) şi `RTOD()` (din radiani în grade).

Pentru funcţia `DTOR()`, expresia numerică primită ca argument reprezintă unghiul, exprimat în grade, a cărui conversie în radiani o dorim, iar pentru funcţia `RTOD()` expresia primită ca parametru reprezintă unghiul, exprimat în radiani, ce se va transforma în grade.

Constanta Π (Pi=3.141592) este obţinută cu ajutorul funcţiei FoxPro `PI()`.

Exemplu

```
? DTOR(90)=PI()/2
.T.
? RTOD (PI()/4)
45.00
```

Funcţiile trigonometrice sinus, cosinus şi tangentă sunt implementate prin funcţiile FoxPro `SIN()`, `COS()` şi `TAN()`, iar funcţiile trigonometrice inverse, arcsin, arccos şi arctangentă, prin `ASIN()`, `ACOS()` şi `ATAN()`. Pentru acestea din urmă, valoarea numerică transmisă ca argument trebuie să se încadreze în anumite intervale, date de definiţiile matematice ale funcţiilor respective.

Exemplu

```
? SIN (PI()/2)
1.00
? COS (PI())
-1.00
? TAN (PI()/4)
1.00
? ASIN (1)
1.57
? ACOS (1)
0.00
? RTOD (ATAN(1))
45.00
? ASIN(SIN(PI()/2))=PI()/2
.T.
```

Acestea reprezintă numai câteva dintre funcţiile FoxPro de manipulare a valorilor numerice, suficiente însă pentru realizarea celor mai diverse calcule matematice.

Tipul şir de caractere

Prezentare generală

Un şir de caractere reprezintă o mulţime ordonată de caractere care se tratează ca un tot unitar. Memorarea fiecărui caracter al unui şir se face într-un octet. Numărul caracterelor dintr-un şir reprezintă lungimea şirului. Un subşir al şirului dat este o porţiune din şir, începând de la o poziţie specificată şi având o lungime dată.

Datele de acest tip au o lungime fixă, care poate ajunge până la maximum 254 de caractere. Cu alte cuvinte, dacă într-un câmp al unei tabele dorim memorarea unor şiruri de caractere, în cazul în care lungimea acestora nu variază foarte mult de la o înregistrare la alta şi dacă lungimea dorită nu depăşeşte 254, vom folosi acest tip de date. Dacă însă şirurile de memorat au o lungime mai mare de 254 sau dacă ele variază foarte mult de la înregistrare la înregistrare se va folosi tipul de date „memo", de asemenea specializat pentru memorarea textelor.

Constantele de tip şir de caractere se specifică prin mulţimea caracterelor care le compun, încadrate între apostrofuri simple sau duble (la ambele capete trebuie să avem acelaşi tip de apostrof). De exemplu, `'Salut'` şi `"Salut"` reprezintă acelaşi şir de caractere. Construcţiile de forma `'Salut"` sau `"Salut'` sunt incorecte.

Pentru a include unul din cele două delimitatoare într-un şir de caractere, mulţimea caracterelor ce alcătuiesc şirul va fi încadrată între delimitatorii de celălalt tip decât cel din şir. Astfel, construcţiile `"10' (zece minute)"` şi `'10" (zece secunde)'` sunt corecte.

Dacă lungimea şirului de caractere este 1, acesta se reduce la un caracter, drept exemple având `'A'`, `"1"`, `'<'`. Dacă lungimea şirului este 0 obţinem şirul nul, sau vid, care se specifică prin două apostrofuri consecutive fără spaţii sau alte caractere între ele, adică `""` sau `''`.

Operanzii care intră în componenţa expresiilor de tip şir de caractere pot fi câmpuri ale unei tabele, funcţii ce returnează şiruri de caractere, variabile sau constante.

Asupra şirurilor de caractere se aplică două tipuri de operatori:

- operatori de concatenare;
- operatori de comparare sau relaţionali.

Operatorii de concatenare sunt doi la număr:

- operatorul de concatenare simplu, „+";
- operatorul de concatenare special, „-".

Operatorul de concatenare simplu face ca din două şiruri de caractere să se obţină un al treilea, prin alipirea celui de-al doilea şir la sfârşitul primului. De exemplu, expresia:

```
'strada '+'George Cosbuc'
```

după evaluare va avea valoarea:

```
'strada George Cosbuc'
```

Operatorul de concatenare special este asemănător cu operatorul de concatenare simplu, cu deosebirea că blancurile de la sfârşitul primului şir sunt trecute la sfârşitul şirului al doilea.

Astfel, din expresia:

```
'Salut     '-' prieteni !'
```

vom obţine după evaluare şirul de caractere

```
'Salut prieteni !     '
```

Se observă că blancurile de la începutul şirului al doilea îşi păstrează poziţia în şir.

Un grup aparte de operatori asupra şirurilor de caractere îl constituie operatorii relaţionali. Aceştia sunt operatori binari ce testează dacă două şiruri de caractere se află sau nu într-o relaţie dată, specifică operatorului (relaţie de incluziune, mai mic sau egal etc.). Rezultatul unei asemenea comparaţii este de tip logic (*adevărat* sau *fals*).

Operatorii relaţionali ce se aplică asupra a două şiruri de caractere sunt prezentaţi în următorul tabel:

Operator	Relaţie
$	inclus în
<	mai mic decât
>	mai mare decât
<>, #, !=	diferit de
<=	mai mic sau egal
>=	mai mare sau egal
==	identic

Aceşti operatori vor fi analizaţi în cele ce urmează.

Operatorul „inclus în", $, returnează *adevărat* dacă primul şir de caractere este conţinut în cel de-al doilea; altfel, returnează *fals*.

Exemplu

De exemplu, expresia:

```
'calcul' $ 'calculator'
```

este adevărată, *pe când expresia:*

```
'calcule' $ 'calculator'
```

va fi evaluată la valoarea logică fals.

Operatorul „mai mic decât" returnează *adevărat* dacă primul şir de caractere este mai mic decât cel de-al doilea. Compararea a două şiruri se face astfel: se ia primul caracter din fiecare şir şi se compară prin intermediul codurilor ASCII corespunzătoare. Dacă primul caracter al primului şir are codul ASCII mai mic decât primul caracter al celui de-al doilea şir, atunci primul şir este mai mic decât cel de-al doilea. În cazul în care codul ASCII corespunzător primului caracter al primului şir este mai mare decât codul ASCII al primului caracter al celui de-al doilea şir, şirul al doilea este mai mic. În caz de egalitate între cele două coduri ASCII se trece la compararea codurilor ASCII ale caracterelor de pe poziţia a doua a fiecărui şir, urmându-se acelaşi algoritm.

Dacă lungimile celor două şiruri diferă, se poate considera că şirul mai mic este completat cu caracterul cu codul ASCII 0 până la egalizarea lungimilor. Deci, în cadrul acestui algoritm, caracterele de pe o poziţie dată în cadrul şirurilor de comparat sunt luate în considerare numai dacă toate caracterele de pe poziţiile anterioare sunt identice în ambele şiruri. Evaluarea unei expresii de acest tip se încheie la pasul în care compararea caracterelor de pe poziţia curentă conduce la un rezultat fals, raportat la relaţia operatorului din expresie.

Această tehnică de comparare se aplică următorilor operatori: <, >, <>, #, !=, <=, >=, ==.

Compararea a două şiruri de caractere de lungimi diferite este controlată de comanda SET EXACT. În cazul SET EXACT OFF, care este şi opţiunea implicită, două şiruri care sunt identice pe lungimea celui mai scurt sunt considerate egale. Când SET EXACT este ON, pentru ca două şiruri de caractere să fi egale, ele trebuie să coincidă caracter cu caracter şi să aibă aceeaşi lungime.

Prelucrarea şirurilor de caractere

FoxPro conţine o multitudine de funcţii pentru prelucrarea şirurilor de caractere, o parte dintre acestea fiind prezentate în continuare.

Funcţii ce returnează informaţii despre şirurile de caractere

Una dintre cele mai folosite funcţii asupra şirurilor de caractere este funcţia `LEN()`, care primeşte ca parametru un şir de caractere şi returnează lungimea acestuia.

Exemplu

```
? LEN('Salutari !')
10
? LEN('Strada George Cosbuc'+' nr.150')
27
```

Un alt tip de informaţii pe care le putem obţine despre un şir de caractere se referă la componenţa acestuia, la caracterele care îl alcătuiesc. În FoxPro, aceste informaţii se obţin cu funcţiile `ISALPHA()`, `ISDIGIT()`, `ISLOWER()` şi `ISUPPER()`:

- `ISALPHA()` returnează *adevărat* dacă şirul de caractere începe cu un caracter alfabetic, altfel returnează *fals*;
- `ISDIGIT()` returnează *adevărat* dacă şirul începe cu o cifră, altfel returnează *fals*;
- `ISLOWER()` returnează *adevărat* dacă şirul începe cu o literă mică, altfel returnează *fals*;
- `ISUPPER()` returnează *adevărat* dacă şirul începe cu o majusculă, altfel returnează *fals*.

Observăm că restul caracterelor din şir sunt ignorate.

Exemplu

```
? ISALPHA ('FoxPro sub DOS')
.T.
? ISDIGIT ('123')
.T.
```

Funcţii referitoare la codificarea caracterelor din şiruri

Una dintre funcţiile cel mai des folosite este funcţia `CHR()`, care returnează un caracter ASCII corespunzător codului numeric transmis funcţiei ca parametru. Această funcţie se foloseşte foarte des la trimiterea de coduri la imprimantă şi la manipularea unor caractere speciale prin codurile acestora.

Exemplu

```
? CHR (49)
1
? CHR(65)=='A'
.T.
```

Pentru a obţine efectul invers, adică aflarea codului unui caracter dat, se foloseşte funcţia `ASC()`. Ea primeşte ca parametru un şir de caractere şi returnează un rezultat numeric reprezentând codul ASCII al primului caracter din şirul dat.

Exemplu

```
? ASC ('A')
65
? ASC('a')=ASC('alfa')
.T.
```

Funcţiile `CHR()` şi `ASC()` sunt funcţii inverse, următorul exemplu demonstrând această afirmaţie.

Exemplu

```
? 'A'==CHR(ASC('A'))
.T.
? 65=ASC(CHR(65))
.T.
```

Funcţii referitoare la subşirurile de caractere

Una dintre principalele caracteristici ale unui şir de caractere este posibilitatea de a se opera asupra subşirurilor acestuia. FoxPro are implementate o mulţime de funcţii pentru manipularea subşirurilor unui şir de caractere, operaţii ca extragerea unui subşir dintr-un şir, testarea incluziunii unui şir în alt şir, numărarea apariţiilor unui şir în alt şir găsindu-şi rezolvarea simplă în acest limbaj.

Extragerea unui subşir dintr-un şir de caractere se realizează cu funcţia `SUBSTR()`. Aceasta primeşte trei argumente, cu următoarele semnificaţii:

- primul reprezintă şirul de caractere din care se extrage subşirul;
- al doilea este un număr reprezentând poziţia de la care se începe extragerea (se extrage inclusiv caracterul respectiv);
- ultima valoare numerică indică numărul de caractere ce se extrag (adică lungimea subşirului). Dacă ea lipseşte, subşirul se întinde până la sfârşitul şirului de bază.

Exemplu

```
? SUBSTR ('ABCDEF', 2, 3)
BCD
? SUBSTR ('Ziua Buna', 6)
Buna
```

În anumite situaţii se pot folosi alte două variante ale acestei funcţii, care au fost introduse pentru uşurarea muncii de programare, acelaşi efect putându-se obţine cu funcţia `SUBSTR()`. Acestea sunt `LEFT()` şi `RIGHT()`, care returnează un subşir al unui şir dat, poziţionat la stânga, respectiv la dreapta acestuia. Primul argument reprezintă şirul din care se extrage, iar cel de-al doilea lungimea subşirului extras.

Exemplu

```
? LEFT ('La multi ani !', 2)
La
? RIGHT ('Noapte buna !', 6)
buna !
```

Au loc echivalenţele:

`LEFT (<şir>, <număr>)` cu `SUBSTR (<şir>, 1, <număr>)`

`RIGHT (<şir>,<număr>)` cu `SUBSTR (<şir>, LEN(<şir>)-<număr>+1)`

Crearea unui şir de caractere dintr-un alt şir, prin repetarea acestuia de un număr dat de ori, se realizează folosind funcţiile `REPLICATE()` şi `SPACE()`. Prima dintre ele returnează un şir de caractere obţinut prin repetarea şirului primit ca prim argument de un număr de ori egal cu cel de-al doilea argument.

Cea de-a doua funcţie, `SPACE()`, reprezintă o formă particulară a funcţiei `REPLICATE()`, ea returnând un şir de spaţii (`CHR(32)`) cu lungimea dată de parametrul primit ca argument.

`SPACE(<număr>)` este echivalent cu `REPLICATE(' ',<număr>)`

Exemplu

```
? REPLICATE ('a ', 5)
a a a a a
? REPLICATE(' ',6)==SPACE(6)
.T.
```

O altă categorie de funcţii care au ca obiect de prelucrare un subşir de caractere al unui şir dat o reprezintă funcţiile `ALLTRIM()`, `LTRIM()`, `TRIM()` şi `RTRIM()`. Aceste funcţii elimină spaţiile de la capetele unui şir de caractere, obţinându-se un subşir al şirului iniţial, ce conţine doar informaţia utilă a şirului. În general, aceste funcţii se folosesc pentru eliminarea spaţiilor introduse de utilizator, în vederea unei mai eficiente folosiri a memoriei sau pentru o mai bună formatare pe ecran.

Funcţiile au următoarele utilizări:

- `ALLTRIM()` elimină spaţiile de la începutul şi sfârşitul şirului de caractere, deci de la stânga şi de la dreapta acestuia;
- `LTRIM()` elimină spaţiile de la începutul şirului, deci de la stânga lui;
- `TRIM()` şi `RTRIM()`, care sunt identice, elimină spaţiile de la sfârşitul şirului transmis, deci de la dreapta acestuia.

Exemplu

```
? ALLTRIM('    GAMA ')=='GAMA'
.T.
? 'Ma numesc '+RTRIM('Ionescu      ')+' Daniel'
Ma numesc Ionescu Daniel
?' si am '+LTRIM('      24')+' ani.'
 si am 24 ani.
```

Efectul invers, adică adăugarea de spaţii sau alte caractere la un şir, la dreapta sau la stânga acestuia, în scopul de a ajunge la o lungime dată a şirului, se obţine cu ajutorul funcţiilor `PADC()`, `PADL()` şi `PADR()`.

Aceste funcţii adaugă la primul argument (transformat în şir de caractere, dacă nu este deja) un alt şir de caractere, la dreapta pentru `PADR()`, la stânga pentru `PADL()` şi la ambele capete pentru `PADC()`, până se obţine o lungime dată a şirului rezultat. Lungimea finală a şirului este precizată prin cel de-al doilea argument transmis funcţiilor, iar şirul de caractere cu care se face completarea ocupă poziţia a treia în lista parametrilor transmişi.

Exemplu

```
? PADL('Pagina 1',40,'-')
--------------------------------Pagina 1
? PADR('Pagina 1',40,'-')
Pagina 1--------------------------------
? PADC('Pagina 1',40,'-')
----------------Pagina 1----------------
```

Căutarea unui subşir într-un şir dat se poate realiza cu ajutorul mai multor funcţii FoxPro, aici prezentându-se doar câteva dintre ele. Funcţia `AT()` primeşte trei argumente cu următoarele semnificaţii: primul reprezintă subşirul căutat, cel de-al doilea şirul în care se caută, iar cel de-al treilea indică a câta apariţie a subşirului respectiv se caută. Rezultatul funcţiei este numeric şi indică poziţia în cadrul şirului de bază în care s-a găsit subşirul căutat (0 în cazul insuccesului).

Funcţia `ATC()` este asemănătoare cu `AT()`, cu diferenţa că are loc o căutare insensibilă la tipul literelor, mici sau mari.

Exemplu

```
? AT ('nr.', 'Strada George Cosbuc,nr.63-64')
22
? ATC ('NR', 'Strada George Cosbuc,nr.63-64')
22
```

Un şir de caractere se poate întinde pe mai multe linii, trecerea la linie nouă fiind indicată de prezenţa în şir a combinaţiei de caractere `CHR(13)+CHR(10)`.

Funcţiile `ATLINE()` şi `ATCLINE()` caută şirul de caractere primit ca prim argument în şirul de caractere care reprezintă al doilea argument. În caz de reuşită, funcţiile returnează numărul liniei în care a fost găsit subşirul, iar în caz contrar, funcţiile returnează 0. Prima funcţie este sensibilă la tipul literelor, pe când cea de-a doua nu.

Căutarea unui subşir într-un şir dat este realizată şi de funcţia `OCCURS()`, dar, spre deosebire de funcţiile anterioare, aceasta returnează numărul de apariţii ale subşirului în şirul dat.

Exemplu

```
? OCCURS ('a', 'ambasada')
4
```

Funcţii care realizează transformări ale şirurilor de caractere

Diferenţierea dintre caracterele alfabetice mici şi mari a dus la necesitatea transformării literelor mici în majuscule şi invers. Aceste operaţii sunt realizate de funcţiile `LOWER()`, `UPPER()` şi `PROPER()`, care au ca efect prelucrarea unui şir de caractere primit ca parametru, astfel:

- `LOWER()` transformă toate majusculele în litere mici, restul caracterelor rămânând neschimbate;
- `UPPER()` transformă toate caracterele mici în majusculele corespunzătoare, restul caracterelor din şir rămânând neschimbate;
- `PROPER()` transformă primul caracter dintr-un cuvânt în majusculă (dacă este alfabetic), iar următoarele în litere mici.

Exemplu

Un exemplu de utilizare a acestor funcţii îl reprezintă compararea a două şiruri de caractere, independent de tipul caracterelor alfabetice care îl compun. Astfel:

```
a='ALFA'
b='alfa'
? UPPER(a)==UPPER(b)
.T.
```

Aceeaşi comparaţie se poate face cu construcţia:

```
? LOWER(a)==LOWER(b)
.T.
```

Ambele şiruri de caractere sunt transformate mai întâi în majuscule, respectiv în litere mici, după care se face compararea.

Exemplu

```
PROPER ('IoNeScu dAniel')
Ionescu Daniel
```

Un tip special de prelucrare a unui şir de caractere este realizat de funcţia `CHRTRAN()`, care dă posibilitatea utilizatorului de a transforma caracterele dintr-un şir după cum doreşte (nu numai de la litere mici la majuscule şi invers). Această funcţie primeşte ca argumente trei şiruri de caractere: primul reprezintă şirul de bază, de la care se pleacă, iar celelalte două sunt folosite pentru codificarea modului de transformare a caracterelor din şirul de bază.

Transformarea are loc astfel: se ia primul caracter din şirul de bază şi se caută în primul şir folosit pentru codificare; dacă se găseşte, caracterul respectiv se înlocuieşte

în şirul de bază cu caracterul de pe aceeaşi poziţie din cel de-al doilea şir folosit la codificare; dacă nu se găseşte, caracterul rămâne neschimbat. În cazul în care caracterul găsit în primul şir de codificare nu are corespondent în cel de-al doilea şir de codificare, acest caracter se elimină din şirul de bază. Apoi se trece la următorul caracter, procedându-se în mod analog.

Exemplu

```
? CHRTRAN ('bomba', 'ba', 'pe')
pompe
? CHRTRAN ('compuse', 'mpse', 'nfz')
confuz
```

O altă funcţie, asemănătoare ca utilizare şi destinaţie cu cea anterioară, este funcţia `STUFF()`. Aceasta înlocuieşte într-un şir de caractere un subşir al acestuia cu un alt şir de caractere. Identificarea subşirului de înlocuit în cadrul şirului de bază se face ca la funcţia `SUBSTR()`: primul argument reprezintă şirul de bază, următorul reprezintă poziţia de la care începe subşirul, iar cel de-al treilea reprezintă lungimea subşirului. În plus faţă de `SUBSTR()`, `STUFF()` primeşte un al patrulea argument, care reprezintă şirul ce va înlocui subşirul respectiv.

Exemplu

```
STORE 'pala' TO sir
sir=STUFF (sir, 3, 0, 'rale')
? sir
paralela
sir=STUFF (sir, 3, 3, 'sar')
? sir
pasarela
sir=STUFF (sir, 7, 2, '')
? sir
pasare
```

Tipuri de date pentru gestiunea timpului

Pentru gestiunea timpului, în Visual FoxPro sunt implementate două tipuri de date:

- unul pentru memorarea datelor calendaristice (cu precizia maximă de o zi);
- altul pentru memorarea atât a datei calendaristice, cât şi a momentului de timp din cadrul zilei respective (ajungându-se la o precizie de sutimi de secundă).

Tipul dată calendaristică

Prin intermediul acestui tip de date sunt gestionate datele calendaristice cuprinse între 1 ianuarie 100 şi 31 decembrie 9999. Datele de acest tip ocupă 8 octeţi: patru pentru an, doi pentru lună şi doi pentru zi (în această ordine). Specificarea unei date de tip dată calendaristică se face prin plasarea lunii, zilei şi anului între paranteze acolade şi separarea lor prin caracterul '/'.

Exemplu

```
STORE {11/24/91} TO data_c
? data_c
11/24/91
```

Ordinea de specificare a zilei, lunii şi anului, modul de scriere a anului (cu două sau patru cifre) şi separatorul dintre cele trei componente ale datei sunt controlate de comenzi ce vor fi prezentate în acest paragraf. Implicit anul se specifică prin două cifre, ordinea implicită este `lună/zi/an`, iar separatorul implicit este '/', adică formatul american de dată calendaristică.

Data calendaristică vidă se specifică prin spaţii în poziţia zilei, a lunii şi a anului, sau printr-un singur spaţiu încadrat de paranteze acolade:

Exemplu

```
data_v={  /  /  }
? data_v={ }
.T.
```

Obs *FoxPro tratează datele invalide (care nu există) ca date calendaristice vide. Astfel:*

```
? {02/330/98}={ }
.T.
```

Asupra datelor calendaristice se pot aplica operatorii „+" şi „-", semnificând avansul, respectiv reculul, cu un număr de zile.

Exemplu

```
? {02/29/99}+1
03/01/99
? {01/01/99}-1
12/31/98
? {02/01/99}-{01/01/99}
31
```

Observăm că adunarea unei zile la o dată calendaristică nu are ca efect creşterea cu 1 a numărului de zile, ci avansarea datei cu o zi în ordine calendaristică. Operatorii relaţionali se pot aplica şi asupra datelor calendaristice. O dată calendaristică este mai mare decât alta atunci când îi urmează, calendaristic.

Formatul de specificare a datelor calendaristice este controlat de comanda SET DATE, care poate primi unul dintre următoarele argumente:

Tipul datei	Formatul
AMERICAN	ll/zz/aa
ANSI	aa.ll.zz
BRITISH	zz/ll/aa
FRENCH	zz/ll/aa
GERMAN	zz.ll.aa
ITALIAN	zz-ll-aa
JAPAN	aa/ll/zz
USA	ll-zz-aa
MDY	ll/zz/aa
DMY	zz/ll/aa
YMD	aa/ll/zz
SHORT	format dat de Windows (cel scurt)
LONG	format dat de Windows (cel lung)

Formatul implicit pentru data calendaristică este cel AMERICAN.

În specificarea anului se pot folosi două cifre, caz în care se presupune automat că ne referim la secolul XX, sau 4 cifre, când anul este specificat complet. Alegerea între aceste două variante se face cu comanda SET CENTURY, unde cazul ON indică două cifre pentru an, iar OFF stabileşte formatul de patru cifre pentru an.

De asemenea, delimitatorii care separă ziua, luna şi anul din expresia unei constante de tip dată calendaristică se pot modifica prin comanda SET MARK TO (urmată de noul caracter delimitator).

Data curentă a sistemului se obţine folosindu-se funcţia DATE().

Exemplu

```
? DATE ()
03/04/99
SET CENTURY ON
SET MARK TO '.'
? DATE ()
03.04.1999
```

Având o expresie de tip dată calendaristică, putem afla în ce zi din cadrul săptămânii cade acea dată. Acest lucru se realizează cu funcția `DOW()`, care returnează numărul zilei respective: 1 pentru luni, 2 pentru marți și așa mai departe.

Exemplu

```
? DOW ({10/02/1864})
1
```

Ziua în cadrul lunii este returnată de funcția `DAY()`, luna în cadrul anului de funcția `MONTH()`, săptămâna în cadrul anului de funcția `WEEK()`, iar anul unei date calendaristice de funcția `YEAR()`. Toate cele patru funcții returnează valori numerice.

Exemplu

```
? DATE()
03/14/99
? DAY(DATE())
14
? MONTH(DATE())
3
? WEEK(DATE())
11
? YEAR(DATE()+365)
2000
```

Am văzut cum, într-o expresie de tip dată calendaristică, putem avansa sau regresa cu un număr de zile folosind operatorii „+" și „–". Pentru a avansa cu o lună sau mai multe sau pentru a regresa cu un număr de luni, vom folosi funcția `GOMONTH()`. Primul argument al funcției reprezintă data calendaristică de la care se pleacă, iar cel de-al doilea reprezintă numărul de luni cu care se avansează (în cazul unei valori pozitive) sau cu care se regresează în timp (în cazul unei valori negative).

Exemplu

```
? GOMONTH ({02/08/91},3)
05/08/91
```

```
? GOMONTH ({01/31/90},1)
02/28/90
```

Tipul moment de timp

Acest tip de date este nou în Visual FoxPro, el fiind introdus pentru a permite utilizatorilor o mai fină gestiune a timpului. Cu ajutorul său se poate realiza gestiunea momentelor de timp din cadrul unei zile cu o precizie de sutime de secundă (nu numai la nivel de zi, cum era cazul tipului dată calendaristică clasic, singurul existent în FoxPro 2.6).

Memorarea datelor de tip moment de timp se face tot pe 8 octeți, patru pentru data calendaristică şi patru pentru momentul de timp din cadrul zilei respective. Cei patru octeți rezervați pentru memorarea datei calendaristice reprezintă de fapt un întreg cu semnificația numărului de zile scurse de la data de 1 ianuarie 100, iar octeții rezervați pentru memorarea momentului de timp din cadrul zilei respective indică numărul de milisecunde trecute de la miezul nopții (ora 0).

O constantă de tip moment de timp se specifică astfel:

```
{ll/zz/aa oo:mm:ss xM}
```

unde:

- `ll` reprezintă luna, `zz` reprezintă ziua, iar `aa` reprezintă anul; acest format poate varia în funcție de comenzile `SET DATE`, `SET YEAR` etc. (a se vedea paragraful anterior);
- `oo` reprezintă ora în cadrul zilei (de la 0 la 12 sau de la 0 la 24, în funcție de prezența sau absența construcției `AM` sau `PM`), `mm` reprezintă minutele, iar `ss` secundele;
- `xM` poate fi `AM` (AnteMeridian), indicând că ora respectivă este înainte de prânz (ora 12:00), sau `PM` (PostMeridian), când ora se situează după prânz.

Spre deosebire de cazul datelor calendaristice, efectul aplicării operatorilor „+" şi „-" asupra unor date de tip moment de timp este acela de avans şi respectiv de recul cu numărul de secunde respectiv.

Exemplu

```
t={01/01/99 02:02:00 AM}
? t+1
01/01/99 02:02:01 AM
? t-1
01/01/99 02:01:59 AM
```

Formatul de specificare a momentelor de timp este controlat de comanda SET HOURS. În cazul SET HOURS TO 12, ora este afişată în formatul prezentat mai sus, iar în cazul SET HOURS TO 24, ora este afişată în formatul oo:mm:ss, unde oo ia valori de la 0 la 23, iar xM lipseşte.

Afişarea secundelor în acest format este dependentă de comanda SET SECONDS. În cazul ON formatul cuprinde şi secundele, iar în cazul OFF este de forma oo:mm.

Exemplu

```
t={01/01/99 02:02:00 PM}
SET HOURS TO 24
? t
01/01/99 14:02:00
SET SECONDS OFF
? t
01/01/99 14:02
```

Momentul de timp curent este returnat de funcţia DATETIME() sub forma unui şir de caractere. Formatul acestui şir este dependent de comenzile prezentate anterior (SET DATE, SET HOURS etc.). Dacă se doreşte ora curentă, fără data calendaristică curentă, se poate folosi funcţia TIME().

Funcţiile DAY(), WEEK(), MONTH() şi YEAR() se aplică şi asupra datelor de tip moment de timp, ele extrăgând din acestea ziua, săptămâna, luna şi respectiv anul (a se vedea paragraful anterior). Extragerea orei dintr-o dată de tip moment de timp se realizează cu ajutorul funcţiei HOUR(), extragerea minutelor cu funcţia MINUTE(), iar a secundelor cu funcţia SEC().

Exemplu

```
dt=DATETIME()
? dt
03/14/99 03:57:19 PM
? HOUR(dt)
15
? MINUTE(dt)
59
? SEC(dt)
19
```

Un control mai fin al timpului este oferit de funcţia SECONDS(). Aceasta returnează numărul de secunde scurse de la miezul nopţii până în momentul respectiv. Partea întreagă a valorii returnate reprezintă secunde, iar partea fracţionară oferă o precizie de până la 1 milisecundă.

Exemplu

```
? TIME()
16:05:14
? SECONDS()
57915.545
```

Tipuri de date speciale

Tipul „memo"

Acest tip de date se foloseşte pentru prelucrarea textelor, adică a şirurilor de caractere de dimensiuni mari sau variabile. De exemplu, dacă într-un câmp al unei baze de date care memorează datele referitoare la personalul unei unităţi economice trebuie stocată, doar pentru directori, o caracterizare, există riscul ca cele 254 de caractere permise pentru tipul şir de caractere să nu fie suficiente. Şi chiar dacă acest neajuns ar fi depăşit, majoritatea persoanelor din baza de date nu au completat acest câmp (câţi dintre angajaţi sunt directori?). Dacă tipul ales pentru acest câmp ar fi şir de caractere, zona de memorie alocată ar fi aceeaşi indiferent de completarea sau lăsarea necompletată a câmpului respectiv. Aceasta ar conduce la o mare risipă de spaţiu de memorie, neajuns înlăturat prin folosirea tipului „memo".

Dar pentru că acum nu posedăm cunoştinţele necesare abordării acestui subiect, îl vom trata mai târziu, într-un paragraf special al capitolului „Exploatarea bazelor de date".

Tipul „general"

Alt tip de date ce se poate folosi pentru câmpurile unei tabele este „general". Acest tip de date permite stocarea în tabele a unor elemente create cu ajutorul altor programe, cum ar fi documente, foi de calcul tabelar, imagini etc., utilizându-se tehnologia OLE, de încorporare şi legare a obiectelor.

Mai multe detalii despre acest tip de câmpuri puteţi găsi în paragraful special destinat acestui subiect din capitolul „Tehnici speciale disponibile în Visual FoxPro".

Conversii între tipurile de date

Fiecare tip de date are propriile sale caracteristici, proprii săi operatori specifici. Uneori este necesară prelucrarea unor date de un anumit tip prin mijloace specifice altui tip de date. Să luăm, de exemplu, cazul ordonării prin indexare a unei tabele cu personalul unei unităţi economice, după anul angajării (deci după vechimea în muncă la unitatea respectivă), în cadrul aceluiaşi an ordonarea făcându-se după nume şi prenume (în ordine alfabetică). Pentru rezolvarea acestei probleme, este necesară construirea unei expresii în care să intervină atât o valoare numerică (anul angajării), cât şi una de tip şir de caractere (nume şi prenume). Pentru combinarea acestora, este necesară conversia uneia dintre ele la tipul celeilalte. În cazul nostru, evident, vom transforma anul în şir de caractere şi-l vom concatena apoi cu numele şi prenumele persoanei.

Acesta este un exemplu de conversie de tip, impusă de necesităţi practice. Există nenumărate cazuri similare, motiv pentru care este importantă cunoaşterea modului în care se realizează aceste conversii.

Şir de caractere ↔ număr

Prima conversie de care ne vom ocupa este cea dintre tipul şir de caractere şi cel numeric (în ambele sensuri). Un număr se transformă în şir de caractere cu ajutorul funcţiei `STR()`. Aceasta primeşte ca argumente, în ordine, expresia numerică de transformat, lungimea viitorului şir de caractere şi numărul de zecimale impus şi returnează şirul de caractere corespunzător.

Dacă lungimea şirului este prea mică pentru numărul de transformat, va fi returnat un şir de asteriscuri semnalizând depăşirea numerică. Dacă, din contră, lungimea şirului este prea mare, poziţiile ce nu se ocupă cu cifre se vor umple cu spaţii. Dacă partea fracţionară conţine mai multe cifre decât poziţiile alocate de noi, numărul va fi trunchiat la numărul de cifre impus.

Exemplu

```
? STR(1432.456,12,4)==' 1432.456 '
.T.
? STR (1432.456,3)
***
? STR (1432.456,7,2)
1432.45
```

Efectul opus funcţiei `STR()`, adică trecerea de la un şir de caractere la o valoare numerică, se obţine folosind funcţia `VAL()`. Ea primeşte ca parametru şirul conţinând valoarea numerică şi returnează numărul corespunzător. Şirul de caractere trebuie să

reprezinte un număr, adică să conţină cifre, punctul zecimal, eventual semnul, altfel transformarea va fi eronată.

Exemplu

```
VAL (' 1433.44  ')
1433.44
? VAL ('1A')
1.00
```

Dată calendaristică ↔ moment de timp

Între tipurile dată calendaristică şi moment de timp se pot realiza conversii cu ajutorul funcţiilor DTOT() şi TTOD(). Prima dintre funcţii primeşte ca argument o dată calendaristică şi returnează o dată de tip moment de timp. Deoarece funcţiei DTOT() i se furnizează doar data calendaristică, nu şi momentul de timp din cadrul acesteia, se presupune că este vorba de ora 12:00:00 AM, adică de miezul nopţii (începutul zilei respective).

Conversia inversă este posibilă prin intermediul funcţiei TTOD(). Aceasta returnează data calendaristică din data de tip moment de timp primită ca parametru.

Exemplu

```
? DATE()=TTOD(DATETIME())
.T.
? DATE()
03/14/99
? DTOT(DATE())
03/14/99 12:00:00 AM
```

Dată calendaristică ↔ şir de caractere

O altă conversie de care ne vom ocupa este cea de la dată calendaristică şi moment de timp la şir de caractere şi invers. Transformarea unei date calendaristice într-un şir de caractere se poate realiza cu funcţiile DTOC() şi DTOS(), iar trecerea inversă cu ajutorul funcţiei CTOD(). Funcţia DTOC() returnează un şir de caractere conţinând data calendaristică primită ca parametru.

Exemplu

```
? DTOC({10/02/90})=='10/02/90'
.T.
```

O formă specială a funcţiei `DTOC()` o reprezintă funcţia `DTOS()`. Aceasta este echivalentă cu funcţia `DTOC()`, cu excepţia formatului furnizat la ieşire. În timp ce la `DTOC()` formatul era cel implicit folosit de sistem la afişarea datelor calendaristice, la `DTOS()` şirul de caractere rezultat are următorul format: **aaaallzz**.

Acest format se foloseşte pentru compararea datelor calendaristice, în următorul exemplu punându-se în evidenţă această proprietate a sa.

Exemplu

Compararea a două expresii de tip dată calendaristică se reduce la compararea a două şiruri de caractere în formatul dat de funcţia `DTOS()`.

```
Data1={03/14/90}
Data2={03/15/89}
a=DTOS(Data1)
? a
19900314
b=DTOS(Data2)
? b
19890315
? a > b
.T.
a=DTOC(Data1)
? a
03/14/90
b=DTOC(Data2)
? b
03/15/89
? a > b
.F.
```

Cum prima dată, `Data1`, este mai mare decât a doua, `Data2`, rezultă că din cele două comparaţii prima variantă dă răspunsul corect, cea de-a doua fiind greşită.

Funcţia `CTOD()` transformă şirul de caractere primit ca parametru într-o dată calendaristică.

Moment de timp ↔ şir de caractere

Conversia unei date de tip moment de timp într-una de tip şir de caractere este realizată de funcţia `TTOC()`, iar transformarea inversă de `CTOT()`.

Exemplu

Alcătuirea unei date de tip moment de timp din mai multe şiruri de caractere conţinând elementele `zi`, `luna`, `an`, `ora`, `min`, `sec`, `apm` (variabile de memorie) se face astfel:

```
dt=CTOT(zi+"/"+luna+"/"+an+" "+ora+":"+min+":"+sec+" "+apm)
```

Metoda este des folosită la citirea unei valori de tip moment de timp pe ecran (în câmpuri de editare). Elementele componente (ziua, ora ...) se citesc în şiruri de caractere şi apoi data respectivă se compune ca mai sus.

Funcţii de uz general

În acest paragraf sunt prezentate o serie de funcţii de uz general, care se aplică mai multor tipuri de date. O primă astfel de funcţie este EVALUATE(). Ea evaluează expresia de tip şir de caractere primită ca parametru, returnând rezultatul obţinut în urma evaluării (indiferent de tipul acestuia). Deci, în funcţie de conţinutul şirului de caractere transmis ca parametru, rezultatul returnat de funcţie va fi de tip şir de caractere, numeric, dată calendaristică etc.

Exemplu

```
? EVALUATE ('6*3/2')
9
? EVALUATE (REPLICATE('1',4)+'/11')  && adica 1111/11
101
? DATE ()
03/09/93
? EVALUATE ('DATE()+1')
03/10/93
```

Oriunde este posibil, se preferă înlocuirea macrosubstituţiei reprezentate de operatorul & cu funcţia EVALUATE(), aceasta fiind mult mai rapidă.

Exemplu

```
a=121
nume='a'
? &nume/11
11.00
? EVALUATE (NUME+'/11')
11.00
```

Un alt tip de testare a unei expresii este cel referitor la valoarea vidă, nulă, a acesteia, adică dacă aceasta este vidă sau nu, semnificaţia termenului „vidă" diferind de la un tip de date la altul:

Tipul datei	Semnificaţia termenului „vidă"
şir de caractere	conţine numai spaţii (CHR(32)), caractere nul (CHR(0)), caractere tab (CHR(9)), sfârşit de linie (CHR(13), CHR(10))
numeric	este 0
dată calendaristică şi moment de timp	data vidă ({ })
logic	este *fals*, .F.
memo, general	fără conţinut

Funcţia primeşte ca argument o expresie, pe care o evaluează şi returnează *adevărat*, dacă expresia este vidă, sau *fals*, în caz contrar.

În prezentarea tipurilor de date implementate în FoxPro, am tratat operatorii relaţionali şi, o dată cu aceştia, modul de comparare a două expresii. Compararea depinde de tipul datelor ce intervin în comparaţie.

Exemplu

Astfel:

```
? 120393 > 10394
.T.
? '120393'>' 10394'
.T.
? {12/03/93} > {01/03/94}
.F.
```

Există o categorie de funcţii ce compară între ele mai multe expresii, returnând informaţii referitoare la această comparaţie. În această grupă intră funcţiile MIN(), MAX() şi BETWEEN(). Primele două, MIN() şi MAX(), compară expresiile primite ca argumente, indiferent de numărul şi de tipul lor (toate fiind însă de acelaşi tip). Funcţia MIN() returnează valoarea minimă obţinută după evaluarea expresiilor din listă, iar funcţia MAX() returnează valoarea maximă.

Exemplu

```
? MIN (64, 7*8, 7*9)
56
? DATE ()
03/09/93
```

```
? MAX ( {03/08/93}, DATE()) = DATE()
NOTE testeaza daca am depasit data de 08.03.1993
.T.
? MIN ('abc', 'ABC', '123')
123
```

O altă funcție FoxPro care are ca scop compararea unor expresii este funcția `BETWEEN()`. Ea testează dacă o expresie se încadrează valoric într-un anumit interval. Expresia testată se transmite funcției ca prim parametru, iar limita inferioară și cea superioară a intervalului ca parametrii doi și trei.

Funcția returnează valoarea `.T.` dacă prima valoare se încadrează în intervalul specificat (considerat închis), altfel returnează `.F.`

Exemplu

```
BETWEEN (144, 100, 200)
.T.
? DATE ()
05/09/93
BETWEEN (DATE(), {05/01/93}, {05/31/93})
.T.
alfa='a1'
? BETWEEN (alfa, 'aa', 'az')
.F.
```

În acest exemplu se compară valoarea variabilei `alfa`, *adică șirul de caractere* `'a1'`, *cu șirurile* `'aa'` *și* `'az'`, *deci se testează dacă este adevărată următoarea dublă comparație:*

```
'aa' <= 'a1' <= 'az'
```

Rezultatul acesteia este fals (`.F.`) (a se vedea compararea șirurilor de caractere, în paragraful „Tipul șir de caractere").

Apartenența la o mulțime este testată în FoxPro prin funcția `INLIST()`. Ea returnează *adevărat* dacă expresia transmisă ca prim parametru se găsește (valoric) printre celelalte expresii transmise. Altfel, returnează *fals*. Toate expresiile din listă trebuie să fie de același tip.

Exemplu

```
zi='duminica'
? INLIST (zi, 'luni', 'marti', 'miercuri', 'joi',;
    'vineri', 'sambata', 'duminica')
.T.
? INLIST (7, 0, 2, 4, 6, 8, 10)
.F.
```

Partea a-II a – BAZE DE DATE

Construirea bazelor de date relaţionale

Capitolul 3

- Componentele unei baze de date relaţionale
- Construirea unei baze de date clasice
  - Construirea tabelelor simple
  - Indexarea tabelelor
  - Baze de date relaţionale clasice. Relaţii între tabele
- Construirea bazelor de date noi
  - Ce aduce nou o bază de date nouă faţă de una clasică?
  - Crearea unei baze de date noi
  - Caracteristicile noi ale tabelelor legate
  - Facilităţi noi la nivelul bazelor de date în ansamblu
  - Vederi
- Elemente de proiectare a bazelor de date relaţionale. Tehnica normalizării tabelelor

Componentele unei baze de date relaţionale

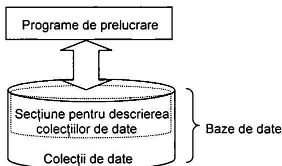
Def

*O **bază de date relaţională** reprezintă o structură folosită la memorarea şi gestionarea datelor descriind un anumit tip de obiecte.*

În primele versiuni de SGBD „relaţionale", bazele de date reprezentau simple tabele, alcătuite din câmpuri şi înregistrări. O dată cu evoluţia acestor tipuri de sisteme, a crescut numărul de tabele gestionate simultan şi au apărut noi tehnici de stabilire a legăturilor între datele din tabele. De exemplu, în versiunile FoxPro 2.6 pentru DOS şi pentru Windows, numărul tabelelor ce pot fi deschise simultan a ajuns la 225, iar legăturile între tabele se realizează prin comenzi introduse în programele de prelucrare.

O bază de date relaţională construită în aceste sisteme este alcătuită din mai multe tabele simple, între care se stabilesc legături, dinamic, la rularea programelor de prelucrare. O bază de date relaţională nu poate fi văzută ca un tot unitar decât prin intermediul programelor de prelucrare, deoarece numai acolo se precizează cum sunt legate între ele tabelele componente. De exemplu, copiind fişierele de date ale unei baze de date relaţionale dintr-un sistem de calcul în altul, nu putem spune că am copiat baza de date la celălalt sistem de calcul, deoarece îi lipsesc relaţiile acesteia. Prin urmare, baza de date relaţională este dependentă de programele de prelucrare.

În Visual FoxPro există posibilitatea definirii statice a relaţiilor între tabele, independent de programele de prelucrare. Mecanismul prin care se realizează acest lucru este următorul: o bază de date relaţională are asociat un fişier special (cu extensia .DBC), în care sunt memorate date referitoare la baza de date în ansamblul său, cum ar fi: tabelele componente, relaţiile permanente între tabele, dicţionarul de date asociat bazei de date etc. În acest fel, între date şi programele de prelucrare se interpune un nou nivel, care asigură independenţa bazelor de date faţă de programele de prelucrare. Acest nivel este asemănător nivelului structurii tabelelor (care realiza independenţa datelor din tabele faţă de programele de prelucrare), numai că este asociat bazei de date relaţionale în ansamblul său. Un program care prelucrează o bază de date relaţională mai întâi citeşte datele care descriu baza de date din fişierul asociat acesteia şi numai apoi, cunoscându-i structura complexă, procedează la prelucrarea datelor respective.



Pe lângă tabelele simple, componentele acestora şi relațiile între ele, o bază de date relațională construită în Visual FoxPro mai poate conține şi alte elemente, alcătuind împreună aşa-numitul **dicționar de date** asociat bazei de date. Printre alte elemente care pot fi incluse într-o bază de date relațională Visual FoxPro se numără şi următoarele:

- nume lungi ale tabelelor şi ale câmpurilor acestora, prin intermediul cărora se obține o mai bună lizibilitate în manipularea acestor elemente;
- alte caracteristici ale câmpurilor tabelelor, cum ar fi formatul implicit de afişare a datelor, textul folosit ca titlu al câmpului, valorile implicite cu care sunt completate inițial câmpurile respective etc.;
- diferite secvențe de cod (proceduri şi funcții) care să fie executate automat la apariția anumitor evenimente, cum ar fi completarea de către utilizator a unui câmp cu date, adăugarea unei noi înregistrări la tabelă, modificarea unei anumite valori dintr-un câmp etc.;
- restricții de integritate, sau condiții ce trebuie îndeplinite (respectate) la modificarea datelor din bazele de date etc.;
- vederi ale bazei de date, acestea reprezentând un fel de tabele construite pe baza şi cu datele tabelelor bazei de date;
- conexiuni cu alte surse de date, precum baze de date create cu alte sisteme de gestiune a bazelor de date, programe de calcul tabelar etc.

Aceste elemente vor fi prezentate în paragrafele următoare ale acestui capitol.

O dată cu introducerea în Visual FoxPro a noului concept de bază de date, o tabelă clasică a fost îmbunătățită cu o serie de caracteristici suplimentare, care sunt disponibile numai dacă tabela respectivă este inclusă într-o bază de date. Prin urmare,

vom avea două tipuri de tabele: cele clasice, simple, şi cele incluse în bazele de date, pe care le vom numi „legate".

În continuare, va fi prezentat modul de construire a tabelelor simple şi a bazelor de date clasice. Apoi va fi prezentată modalitatea de construire a bazelor de date noi şi deci a tabelelor legate.

Construirea bazelor de date clasice

Construirea tabelelor simple

Def

*O **tabelă** reprezintă o structură în care se pot memora date descriind un anumit tip de elemente. Fiecare caracteristică a elementelor alcătuieşte un **câmp**, iar elementele propriu-zise sunt memorate în **înregistrări**.*

Structura tabelară este pusă în evidenţă în următoarea figură: câmpurile reprezintă coloanele tabelei, iar înregistrările liniile acesteia.

TABELA.DBF					
	Câmp 1	Câmp 2	Câmp 3	...	Câmp m
Înregistrare 1	...	...	...	...	...
Înregistrare 2	...	...	...	...	...
...	...	...	...	...	...
Înregistrare N	...	...	...	...	...

De exemplu, tabela persoanelor angajate la o unitate economică ar putea avea drept câmpuri: numele şi prenumele persoanei, seria buletinului, data naşterii, data angajării etc. În fiecare înregistrare a tabelei este memorată o anumită persoană (adică datele despre o anumită persoană). La intersecţia unei linii cu o coloană, deci a unui câmp cu o înregistrare, se găseşte o anumită caracteristică a unei anumite persoane.

Definirea unei tabele implică specificarea numelui tabelei respective, a câmpurilor componente, împreună cu caracteristicile acestora (nume, tip, lungime etc.), şi a indecşilor folosiţi la ordonarea datelor din tabelă.

Obs

Indecşii *reprezintă o tehnologie cu ajutorul căreia datele din tabele sunt văzute într-o anumită ordine, dată de un anumit criteriu. Aceştia vor fi trataţi într-un paragraf special din acest capitol.*

Fiecare câmp al unei tabele este caracterizat prin:

- **nume** – care reprezintă identificatorul prin care se face referire la câmpul respectiv, la datele memorate în acesta;
- **tip** – adică tipul datelor care pot fi memorate în câmp. Acesta poate fi: şir de caractere, numeric (de diferite feluri), dată calendaristică sau moment de timp, logic, „memo" sau „general". În paragraful referitor la tipurile de date gestionate în Visual FoxPro sunt descrise toate tipurile de date disponibile;
- **lungime** – adică numărul de caractere pe care îl ocupă fiecare dată memorată în câmpul respectiv;
- **numărul de zecimale** – caracteristică valabilă în cazul câmpurilor numerice, care indică numărul de cifre de după punctul zecimal memorate în câmp. Lungimea câmpului (precizată anterior) trebuie să includă şi această valoare şi punctul zecimal propriu-zis, în cazul câmpurilor de tip numeric simplu;
- **fanionul de indexare** – care indică dacă se stabileşte un index pe baza câmpului respectiv. Poziţionarea acestui indicator la crearea tabelei determină crearea unui index pentru acea tabelă, index cu acelaşi nume cu câmpul respectiv, cheia de indexare fiind, de asemenea, câmpul respectiv;
- **fanionul de valoare nulă** (**NULL**) – arată dacă în câmpul respectiv poate fi memorată sau nu o valoare nulă. Această caracteristică a fost introdusă datorită necesităţii diferenţierii între un câmp lăsat necompletat de utilizator şi unul completat cu valoarea 0.

Exemplu

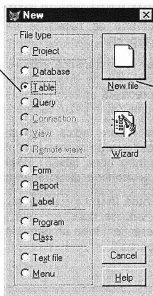
Să presupunem că dorim crearea unei tabele în care să memorăm datele referitoare la angajaţii unei societăţi comerciale. Numele tabelei va fi ANGAJATI.DBF*, iar câmpurile sale vor fi:*

Nr.crt.	*Denumire*	*Tip*	*Lung.,zec.*	*Index*	*NULL*
1	*NUME*	*Şir car.*	*14*	*Nu*	-
2	*PRENUME*	*Şir car.*	*30*	*Nu*	-
3	*COD_NUM_P*	*Şir car.*	*13*	*Da*	-
4	*DATA_NAST*	*Dată cal.*	*8*	*Da*	-
5	*SEXUL*	*Logic*	*1*	*Nu*	-
6	*ADRESA*	*Memo*	*4*	*Nu*	-
7	*TELEFON*	*Şir car.*	*12*	*Nu*	-

8	STUDII	Şir car.	1	Nu	-
9	DATA_ANG	Dată cal.	8	Nu	-
10	FUNCTIA	Şir car.	3	Nu	-
11	DEPARTAM	Şir car.	3	Nu	-
12	SAL_BRUT	Întreg	4	Nu	-

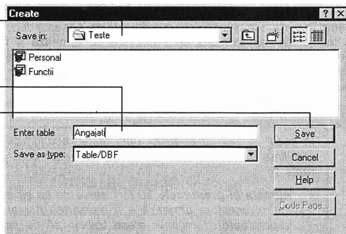
Practic, crearea unei tabele simple debutează cu alegerea opţiunii **New** a submeniului **File**, după care, din fereastra deschisă pe ecran, trebuie ales butonul **Table** (indicându-se astfel că se doreşte crearea unei tabele).

Se selectează acest buton pentru a indica sistemului că se doreşte crearea unei tabele

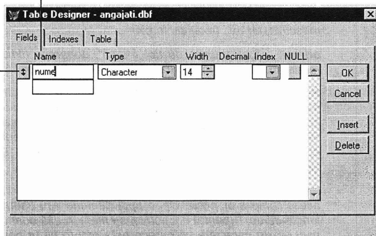


Apoi se acţionează acest buton

Apoi se acţionează butonul **New file**, ceea ce conduce la afişarea pe ecran a unei ferestre în care trebuie specificat numele noii tabele.



După specificarea numelui tabelei ce urmează a fi creată, pe ecran este deschisă fereastra Constructorului de tabele, în care vor fi precizate caracteristicile noii tabele.



La crearea tabelelor, o atenţie deosebită trebuie acordată stabilirii pentru fiecare câmp în parte a tipului de date corespunzător. Aceasta depinde de natura datelor ce vor fi memorate, de tehnicile de prelucrare ce urmează a fi folosite şi de experienţa proiectantului. Mai multe amănunte despre tipurile de date disponibile, despre avantajele şi dezavantajele fiecăruia dintre ele, se găsesc în capitolul special destinat acestui subiect.

Exemplu

În cazul tabelei cu angajaţii unei unităţi economice (ANGAJATI.DBF), pentru stabilirea caracteristicilor câmpurilor s-a ţinut cont de următoarele considerente:

- ***numele şi prenumele*** *au fost alese de tip şir de caractere, deoarece în cadrul acestora urmează a fi memorate numai litere (şi eventual spaţii separatoare). Lungimile alese trebuie să fie suficient de mari pentru a putea cuprinde orice nume ce urmează a fi memorat (în cazul nostru, am ales 14 şi respectiv 30);*
- ***codul numeric personal*** *identifică în mod unic o persoană şi din acest motiv el poate fi folosit pe post de cheie primară a tabelei. Deşi acest câmp conţine numai cifre, s-a preferat alegerea tipului şir de caractere, deoarece fiecare cifră din câmp are o anumită semnificaţie. Extragerea unei cifre dintr-un număr este mai dificilă decât extragerea unui caracter dintr-un şir;*
- ***data naşterii*** *va fi, evident, de tip dată calendaristică;*
- ***sexul****, deoarece conţine doar două valori (masculin sau feminin), a fost declarat de tip logic. S-a făcut convenţia că* .T. *(*adevărat*) reprezintă sexul masculin, iar* .F. *(*fals*) cel feminin;*
- ***adresa*** *a fost stabilită de tip „memo", deoarece dimensiunea sa variază foarte mult de la o înregistrare la alta (de la o persoană la alta);*
- ***telefonul****, ca şi codul numeric personal, ar fi putut fi declarat de tip numeric, deoarece conţine doar cifre. S-a ales totuşi tipul şir de caractere pentru a permite introducerea în câmp a spaţiilor (sau a altor caractere) separatoare, dar şi pentru a menţine cifra 0 la începutul unui număr (de exemplu, memorarea numărului de telefon 092291988 ca dată de tip numeric s-ar face sub forma 92291988);*
- ***studiile*** *pot fi: fără studii, cu studii medii şi cu studii superioare. Pentru acest câmp s-ar putea alege tipul numeric sau tipul şir de caractere, cu lungimea de 1 (pentru economie, se memorează doar un cod). S-a preferat tipul şir de caractere, deoarece este mai lizibil (*F *pentru lipsa studiilor,* M *pentru studii medii şi* S *pentru studii superioare);*
- ***data angajării*** *a fost declarată de tip dată calendaristică;*

- ***funcţia***: *în cadrul unităţii economice există aproximativ 85 de funcţii. Deşi ar fi fost de ajuns 2 caractere, pentru memorarea tuturor codurilor de funcţii existente (dar şi pentru a creşte lizibilitatea codurilor respective) s-au alocat 3 caractere;*
- ***departamentul*** *respectă aceleaşi considerente ca şi câmpul anterior;*
- ***salariul brut*** *s-a stabilit a fi de tip întreg, deoarece acest tip de date oferă suficient spaţiu pentru memorarea chiar şi a celor mai mari salarii existente şi ocupă doar 4 caractere.*

Indexarea tabelelor

Def

Indexarea *reprezintă o tehnică de ordonare logică a datelor dintr-o tabelă, după diferite criterii, operaţie care însă nu afectează ordinea fizică a datelor din tabelă, ci doar modul în care acestea sunt văzute de utilizator.*

Ordonarea unei tabele presupune stabilirea unui criteriu după care să fie parcurse înregistrările ei. Acest criteriu poate fi un câmp sau o combinaţie de câmpuri ale tabelei. De exemplu, tabela angajaţilor unei unităţi economice poate fi ordonată după numele de familie al persoanelor (un singur câmp), după numele complet al acestora (nume şi prenume), după codul numeric personal, dupa salariu etc.

Criteriul de ordonare reprezintă de fapt o expresie, numită în această postură **cheie de ordonare**, în alcătuirea căreia intră câmpuri ale tabelei. Pentru stabilirea ordinii înregistrărilor tabelei se evaluează expresia respectivă pentru fiecare înregistrare în parte. Valorile obţinute se compară între ele, iar ordinea înregistrărilor în tabelă se stabileşte în funcţie de rezultat.

Există două tipuri de ordonări ale unei tabele:

- **fizică**, numită şi **sortare**, ce constă în rearanjarea fizică a datelor din tabelă într-o altă ordine, dată de criteriul respectiv. În urma sortării rezultă o nouă tabelă, cu aceeaşi structură cu cea de bază, dar cu înregistrările aranjate în ordinea dorită;
- **logică**, numită şi **indexare**, care constă în construirea unui fişier special, folosit la regăsirea datelor din tabelă în ordinea dorită. Acest fişier este văzut ca un filtru aplicat tabelei respective, filtru care modifică ordinea înregistrărilor sale. Prin indexare, ordinea fizică a înregistrărilor din tabelă nu se modifică. Se schimbă însă modul în care utilizatorul are acces la datele respective.

De exemplu, accesul la datele din tabela angajaţilor unei unităţi economice, indexată după salariul brut, se face în conformitate cu următoarea schemă:

ANGAJATI.CDX	
Sal_brut	**Poziție**
849000	23
867000	46
956000	6
1216000	1
1453000	32
1679000	2
2100000	9
...	...

ANGAJATI.DBF				
Nr.înr.	**Nume**	**Prenume**	**...**	**Sal_brut**
1	Popa	Mihai	...	1216000
2	Constantinescu	Valentin	...	1679000
3	Dumitrescu	Ioana	...	867000
4	Vasiliu	Mioara	...	3531000
5	Vlad	Alexandru	...	2130000
6	Apopei	Constantin	...	956000
7	Vieriu	Liviu	...	879000
...	...	...	...	...

Se observă că, dacă se dorește citirea datelor persoanei cu al treilea salariu ca mărime din unitatea economică, se procedează după cum urmează:

- se citește din fișierul index asociat înregistrarea a treia, care conține atât salariul brut al persoanei (valoarea cheii pentru înregistrarea respectivă), cât și indicatorul către înregistrarea corespunzătoare din tabela angajaților;
- pe baza acestui indicator se stabilește poziția fizică a înregistrării corespunzătoare din tabelă (în exemplul nostru, înregistrarea a 6-a);
- apoi se citesc datele din înregistrarea tabelei.

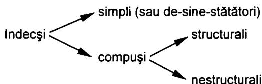
Chiar dacă din tabelă se preiau datele înregistrării a 6-a, utilizatorul nu vede acest lucru. Din cauza indexului, el vede această înregistrare ca fiind pe poziția 3 în tabelă.

Inițial, o tabelă indexată avea asociat un fișier special în care erau memorate informațiile referitoare la ordinea datelor. Ori de câte ori se solicitau date din tabelă, se citeau mai întâi informațiile din fișierul index și, pe baza acestora, se localizau fizic datele din tabelă. Pentru construirea mai multor indecși pentru aceeași tabelă erau necesare mai multe astfel de fișiere index. Pentru a putea fi folosit, fiecare dintre ele trebuia deschis explicit o dată cu deschiderea tabelei.

Versiunile mai noi de FoxPro au fost modernizate pentru a accepta mai mulți indecși într-un singur fișier. Astfel, la deschiderea unei tabele indexate este necesară deschiderea doar a fișierului index asociat (ceea ce se poate face automat), care conține informațiile pentru toți indecșii. Utilizarea acestora se reduce doar la specificarea acelui index care dă ordinea curentă de acces la înregistrări.

Tipurile de indecşi

În FoxPro 2.6 existau două clase de indecşi, cei simpli şi cei compuşi. Primii conţineau un singur criteriu de indexare, pe când cei compuşi conţineau mai multe astfel de criterii. Indecşii compuşi puteau fi, la rândul lor, de două tipuri: structurali şi nestructurali. Indecşii structurali erau memoraţi într-un fişier index cu acelaşi nume cu al tabelei, fişier care era deschis automat o dată cu tabela. Indecşii nestructurali ai unei tabele erau memoraţi în fişiere cu nume diferite de cel al tabelei, iar pentru utilizare trebuia să fie deschişi explicit.



Dintre tipurile de indecşi prezentaţi mai sus, cei mai avantajoşi sunt cei compuşi structurali, care sunt creaţi implicit de Visual FoxPro. Dacă se doreşte crearea unor alte tipuri de indecşi, metoda ce trebuie folosită este cea primară, manuală, adică introducerea comenzilor corespunzătoare în fereastra de comenzi a sistemului.

Exemplu

De exemplu, dacă vrem să creăm pentru tabela ANGAJATI.DBF *un index simplu, pentru accesul la persoane în ordinea codului numeric personal, putem introduce în fereastra de comenzi următoarea secvenţă de instrucţiuni:*

```
USE angajati
INDEX ON cod_num_p TO icodnump
...
```

În cele ce urmează, ne vom ocupa doar de indecşii compuşi structurali, deoarece aceştia sunt creaţi automat în Visual FoxPro, neexistând nici un motiv (afară poate de compatibilitatea cu versiunile anterioare) pentru crearea unor indecşi de alte tipuri.

Aceşti indecşi sunt memoraţi toţi în acelaşi fişier, cu acelaşi nume cu cel al tabelei, extensia fiind însă .CDX. Fiecărui index dintr-un asemenea fişier i se atribuie un nume, numit **etichetă index**, nume prin care se face referire la el în comenzile de prelucrare.

Un index poate fi considerat un filtru aplicat tabelei, filtru prin care datele respective sunt privite într-o anumită ordine. Văzuţi din această perspectivă, indecşii pot chiar anula accesul la unele înregistrări ale tabelei. Un tip special de filtrare prin indexare este acela al înregistrărilor cu aceeaşi valoare a cheii de indexare.

Dacă, de exemplu, indexăm tabela angajaţilor după numele lor de familie, poate apărea situaţia existenţei în tabelă a două persoane cu acelaşi nume de familie. Prin urmare, în tabelă apar două (sau mai multe) înregistrări care corespund aceleiaşi valori a cheii de indexare. Din acest punct de vedere, indecşii pot fi de două feluri:

- **normali** (în engleză *Regular*), care construiesc pentru fiecare înregistrare a tabelei câte o înregistrare în fişierul index respectiv, indiferent de duplicarea (sau multiplicarea) valorii cheii de indexare. Prin urmare, într-o tabelă astfel indexată toate înregistrările sunt accesibile, indiferent de valoarea cheii de indexare;
- **unici** (în engleză *Unique*), care permit o unică valoare a cheii de indexare. Dacă două sau mai multe înregistrări ale tabelei corespund aceleiaşi valori a cheii de indexare, numai prima dintre acestea va fi disponibilă (accesibilă), restul neapărând în tabelă (deşi fizic există).

Legarea tabelelor între ele în cadrul modelului relaţional a impus alte două tipuri de indecşi, cel candidat şi cel cheie:

- indexul **candidat** reprezintă un index asemănător cu cel unic, dar interzice încărcarea de înregistrări care dublează valoarea cheii de indexare. În cazul indecşilor unici se permitea încărcarea acestora, dar înregistrările respective erau ascunse utilizatorului. În cazul indecşilor candidat, când se încearcă încărcarea într-o înregistrare a unor valori care, după evaluarea cheii de indexare, conduc la o valoare care mai există în tabelă (la o altă înregistrare), este generat un mesaj de eroare;
- indexul de tip **cheie primară**. În cadrul unei tabele pot exista mai multe câmpuri (sau, mai precis, mai multe criterii) care să asigure identificarea unică a înregistrărilor tabelei. Dintre acestea se poate stabili unul care să fie folosit drept cheie primară a tabelei, atunci când se creează relaţii între tabela respectivă şi alte tabele ale unei baze de date. Această caracteristică a indexului determină un anumit tip de relaţie între tabele, subiect tratat în paragraful referitor la bazele de date relaţionale.

Tipurile de indecşi prezentate (normali, unici, candidaţi şi de tip cheie primară) impun anumite restricţii de acces, în funcţie de valorile cheii de indexare. Accesul la înregistrările tabelei este din ce în ce mai restrictiv, în ordinea: indecşi normali, unici, candidat şi cheie primară. De fapt, ultimele două tipuri de indecşi se află pe acelaşi nivel de restrictivitate, diferenţa între ele fiind dată de modul de utilizare, de tipul de relaţie ce se doreşte a fi stabilită între tabela respectivă şi o alta.

Responsabilitatea stabilirii tipului de index necesar revine proiectantului bazei de date. De exemplu, pentru tabela ANGAJATI.DBF, stabilirea unui index unic, candidat sau de tip cheie primară după numele de familie al persoanelor reprezintă o alegere greşită, deoarece în oricare dintre aceste cazuri pot apărea situaţii conflictuale. Pentru două persoane cu acelaşi nume de familie:

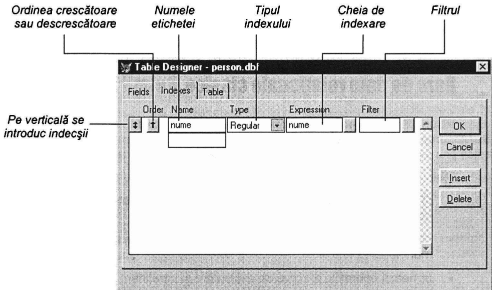
- în cazul indexului unic, nu va fi accesibilă a doua persoană;
- în cazul indecşilor candidat sau cheie primară, nu va putea fi introdusă înregistrarea corespunzătoare celei de-a doua persoane, fiind generată o eroare.

Ca index unic, candidat sau cheie primară ar putea fi ales câmpul cod numeric personal, deoarece acesta identifică în mod unic fiecare înregistrare din tabelă (nu există două persoane cu acelaşi cod numeric personal).

Crearea indecşilor

Să vedem cum pot fi creaţi indecşii pentru o tabelă simplă. La proiectarea unei tabele trebuie stabilite şi eventualele criterii de ordonare, de indexare sau sortare, iar la crearea propriu-zisă a unei tabele trebuie precizaţi indecşii corespunzători.

În fereastra pentru crearea tabelelor, pagina **Fields**, una dintre caracteristicile ce trebuie precizate la adăugarea unui nou câmp este **fanionul de indexare**. Poziţionarea acestuia în starea ↑ sau ↓ determină crearea unei etichete index (în fişierul compus structural) cu acelaşi nume cu al câmpului respectiv, criteriul de ordonare fiind de asemenea câmpul respectiv. Ordinea este crescătoare pentru starea ↑ şi descrescătoare pentru starea ↓.



În figura anterioară se observă cum în pagina **Indexes** (indecşi) a ferestrei Constructorului de tabele a fost introdusă automat eticheta index **nume**, deoarece la adăugarea acestui câmp a fost poziţionat fanionul de indexare în starea ↑.

Fiecare etichetă index introdusă în această fereastră va fi descrisă prin:

- **sensul ordonării**, crescător sau descrescător, care se specifică în coloana **Order** (ordine);
- **numele etichetei index**, care se precizează în coloana **Name** (nume);
- **tipul indexului**, ce se alege din lista **Type** (tip) şi poate fi unul dintre cele patru prezentate mai înainte;
- **cheia de indexare**, introdusă în coloana **Expression** (expresie), cheie care poate fi fie un singur câmp al tabelei, fie o expresie complexă, construită pe baza mai multor câmpuri ale tabelei. În acest din urmă caz, expresia se poate construi mai uşor cu ajutorul Constructorului de expresii, pornit la acţionarea butonului din dreapta coloanei;
- un **filtru** care să restrângă accesul la înregistrările tabelei, pe baza unor criterii impuse de utilizator. Expresia de filtrare, introdusă manual în coloana **Filter** (filtru) sau cu ajutorul Constructorului de expresii apelat la acţionarea butonului din dreapta acestei coloane, este evaluată pentru fiecare înregistrare în parte. Dacă valoarea obţinută este *adevărat*, se va permite accesul utilizatorului la înregistrarea respectivă. În caz contrar, înregistrarea va fi ascunsă utilizatorului.

În cele de mai sus a fost prezentată modalitatea de creare a indecşilor pentru tabele. Mai multe aspecte legate de indecşi (modul lor de utilizare) sunt prezentate în capitolul referitor la exploatarea bazelor de date.

Baze de date relaţionale clasice. Relaţii între tabele

Visual FoxPro menţine compatibilitatea cu versiunile anterioare ale acestui SGBD, drept pentru care se pot construi baze de date complexe şi în maniera clasică, adică prin legarea dinamică a tabelelor, la rularea programelor de prelucrare.

Metoda de construire a unei astfel de baze de date este următoarea:

- mai întâi se deschid, în zone de lucru diferite, tabelele ce urmează a alcătui noua bază de date;
- apoi, între aceste tabele se stabilesc relaţii de tip una la una sau de tip una la mai multe;
- urmează operaţiile efective de exploatare şi întreţinere a bazei de date;

- după ce exploatarea bazei de date s-a terminat, tabelele respective se închid, baza de date fiind astfel „destrămată".

Deoarece operația de construire a unei baze de date clasice se realizează la rularea programelor de prelucrare şi presupune operații de gestiune a tabelelor respective, metoda va fi prezentată în detaliu în capitolul referitor la exploatarea bazelor de date.

Construirea bazelor de date noi

Ce aduce nou o bază de date nouă față de una clasică?



*O **bază de date nouă** reprezintă un „container" în care sunt memorate, printre altele, lista tabelelor componente, o serie de caracteristici noi ale acestora, relațiile permanente între tabele şi secvențe de cod ce urmează a fi lansate în execuție la apariția diferitelor evenimente.*

În vechea accepțiune (din versiunile FoxPro 2.6 pentru DOS şi Windows), o bază de date reprezenta un ansamblu de tabele, legate între ele prin relații construite în programele de prelucrare. Astfel, baza de date nu apărea ca un tot unitar decât prin intermediul programelor de prelucrare; deci era dependentă de acestea. În noua accepțiune (din Visual FoxPro), baza de date se concretizează într-un fişier distinct, în care sunt memorate o serie de caracteristici ale sale. În acest fişier sunt precizate, printre altele, şi tabelele componente şi relațiile stabilite între acestea. Prin urmare, baza de date este acum independentă de programele care o prelucrează, deoarece informațiile privind structura sa sunt memorate într-un fişier distinct (care are extensia implicită .DBC).

În fişierul bazei de date sunt memorate atât caracteristici suplimentare ale tabelelor componente (care altfel nu sunt disponibile), cât şi caracteristici ale bazei de date în ansamblul său. O tabelă inclusă într-o bază de date poate fi îmbunătățită prin introducerea unor facilități suplimentare la nivelul câmpurilor componente sau la nivelul tabelei în ansamblul său.

Unei tabele legate îi poate fi atribuit un nume lung, ce poate creşte substanțial lizibilitatea programelor de prelucrare. La fel şi câmpurilor acestor tabele li se pot atribui astfel de nume. De asemenea, pentru câmpurile unei tabele legate mai pot fi precizate: formatul implicit de afişare a datelor din câmpul respectiv, o machetă folosită la introducerea datelor în câmp, un titlu nou pentru câmpul respectiv, titlu folosit (în antetul listingului) atunci când este afişat conținutul tabelei respective, o valoare inițială implicită a câmpului respectiv, o regulă de validare la nivel de câmp şi, eventual, un mesaj care să fie afişat pe ecran în cazul nerespectării condiției de validare impuse.

La nivelul tabelei în ansamblu, pot fi precizate: o regulă de validare la nivel de înregistrare (la modificarea unei înregistrări) şi, eventual, un mesaj de eroare, în cazul nerespectării acestei reguli, precum şi secvenţe de cod care să fie executate la inserarea unei noi înregistrări, la actualizarea (modificarea) sau la ştergerea uneia existente.

Aceste caracteristici vor fi prezentate mai pe larg în cele ce urmează.

Crearea unei baze de date noi

Construirea unei baze de date de tip nou se realizează astfel:

- mai întâi se construieşte un fişier bază de date nou, în care vor fi memorate datele referitoare la baza de date şi la tabelele componente;
- apoi se încorporează pe rând tabelele simple create anterior sau se construiesc unele noi care să fie direct încorporate în noua bază de date;
- pentru tabelele simple urmează specificarea acelor caracteristici noi, care nu se puteau declara la construirea lor ca tabele izolate;
- apoi se stabilesc legăturile, relaţiile între tabele;
- se pot preciza o serie de caracteristici noi ale bazei de date în ansamblul său;
- de asemenea, se pot construi şi încorpora în baza de date o serie de elemente speciale, precum vederile, conexiunile etc.

Crearea efectivă a fişierului bazei de date se realizează astfel:

- mai întâi se deschide fereastra **New** (nou), prin alegerea opţiunii cu acelaşi nume din submeniul **File**. În această fereastră se alege butonul **Database** (bază de date) şi apoi **New file** (fişier nou);
- pe ecran va fi deschisă o fereastră în care se va specifica numele şi locul pe disc pentru noul fişier. După acţionarea butonului **Save** din această fereastră va fi pornit Constructorul de baze de date, a cărui fereastră arată ca în figura următoare, iar la meniul sistemului se adaugă un nou submeniu, numit **Database** (bază de date), care conţine opţiunile pentru realizarea diferitelor operaţii specifice;

Caracteristici noi ale tabelelor legate

Nume lungi pentru tabele şi câmpurile componente

Unei tabele legate i se poate ataşa un nume lung (de maximum 128 de caractere), care conduce la o mai bună lizibilitate. Una este să desemnăm o tabelă pentru memorarea mijloacelor fixe ale unei unităţi economice prin identificatorul `MIJFIX` şi alta dacă aceasta este denumită `MijloaceFixe`. Facilitatea numelor lungi se extinde şi la nivelul câmpurilor unei tabele legate. Un câmp al unei astfel de tabele poate fi identificat printr-un nume de până la 128 de caractere. De exemplu, tabela mijloacelor fixe ar putea avea câmpuri precum `CodMijlocFix`, `ValoareDeInventar`, `AmortizareLunara` etc.

Obs

Aceste nume nu trebuie să conţină spaţii libere. În plus, sistemul nu face deosebire între literele mari şi cele mici. Dacă o tabelă a fost creată mai întâi ca o tabelă simplă (neinclusă într-o bază de date) şi abia apoi introdusă într-o bază de date, o dată definite numele lungi pentru câmpurile acesteia, cele vechi (de maximum 10 caractere) nu mai pot fi folosite.

Exemplu

O dată tabela `ANGAJATI` *transformată dintr-una simplă într-una legată (inclusă într-o bază de date), câmpurile sale ar putea fi denumite astfel:*

1	NUME	Nume
2	PRENUME	Prenume
3	COD_NUM_P	CodNumericPersonal
4	DATA_NAST	DataNasterii
5	SEXUL	Sex
6	ADRESA	Adresa
7	TELEFON	Telefon
8	STUDII	Studii
9	DATA_ANG	DataAngajarii
10	FUNCTIA	Functie
11	DEPARTAM	Departamentul
12	SAL_BRUT	SalariulBrut

O comandă de afişare a conţinutului tabelei ar putea arăta astfel:

```
LIST FIELDS Nume, Prenume, DataNasterii, DataAngajarii
```

Comentarii şi observaţii referitoare la tabelele legate şi la câmpurile componente

Deşi comentarea tabelelor componente ale unei baze de date şi a câmpurilor acestora nu reprezintă o facilitate puternică, deosebită, este bine să fie folosită ori de câte ori se construieşte o astfel de tabelă, fie pentru o documentare ulterioară, fie pentru a da posibilitatea celor care folosesc tabela să înţeleagă modul de concepere, de proiectare a acesteia.

Formatul de afişare şi citire a câmpurilor tabelelor legate

Pentru fiecare câmp al unei tabele legate poate fi specificat un format de afişare implicit, care să servească la formatarea datelor din câmpul respectiv atunci când acestea sunt afişate într-o fereastră de editare a tabelei (**Browse**) sau într-o listă a acesteia (comanda `LIST` sau un raport, de exemplu).

Formatul de afişare reprezintă de fapt un şir de caractere, alcătuit dintr-o serie de coduri, în funcţie de care sistemul stabileşte modul în care va prezenta datele respective. De exemplu, un format de afişare de tipul `999999999.99` desemnează o valoare numerică afişată cu 9 cifre în faţa punctului zecimal şi două după acesta. Pentru mai multe detalii referitoare la formatul de afişare a datelor se poate consulta paragraful special destinat acestui subiect din capitolul Operaţii de Intrare/Ieşire.

Exemplu

Să considerăm câmpul **`Adresa`** *al tabelei* `PERSONAL`*. Acest câmp este de tip „memo", astfel încât, dacă se doreşte prezentarea conţinutului său cu ajutorul comenzii* `LIST`*, este necesară precizarea unui format de afişare. În lipsa acestuia, ca urmare a comenzii* `LIST`*, respectivul câmp va fi afişat într-un format de tipul:*

```
Record#  NUME              ADRESA

      1  Toma              Memo
      2  Popescu           Memo
    ...  ...               ...
```

Observăm în lista de mai sus cum câmpul **`Adresa`** *nu este explicitat, pe poziţia sa în listing apărând cuvântul „Memo". Dacă însă pentru acest câmp specificăm un format de afişare de tipul:*

```
MLINE(Adresa,1)
```

(care extrage prima linie a câmpului „memo"), atunci afişarea cu comanda **LIST** *va avea ca rezultat scrierea pe poziţia câmpului* **Adresa** *a primei linii a câmpului respectiv (pentru detalii a se consulta paragraful referitor la câmpurile „memo" în capitolul „Exploatarea bazelor de date").*

Record#	NUME	ADRESA
1	Toma	Strada Cuza Voda, nr.154
2	Popescu	B-dul N.Balcescu, nr.14
...	...	...

Tot legat de afişarea datelor din câmpurile unei tabele legate, pentru fiecare câmp în parte se poate preciza un text care să fie folosit în antetul listingului tabelei, ca denumire a câmpului respectiv. Textul poate conţine şi spaţii, spre deosebire de denumirea efectivă a câmpului, care nu poate conţine acest caracter.

Exemplu

De exemplu, pentru tabela **PERSONAL**, *câmpul* **Data_ang**, *am putea preciza un text precum* **Data angajarii**.

Pe lângă formatul de afişare, pentru un câmp al unei tabele legate mai poate fi precizat un format de citire a datelor în câmpul respectiv sau, altfel spus, o mască pentru a restricţiona introducerea datelor în câmp. O astfel de mască este asemănătoare formatului de afişare a datelor, dar se referă la încărcarea acestora în tabelă.

De exemplu, un format de tipul !AAXXXXXXXXX restrânge introducerea datelor în câmpul respectiv doar la acele coduri care încep cu o literă (care este transformată automat în majusculă), după care urmează alte două litere şi în continuare alte caractere (litere, cifre sau alte semne), până la lungimea maximă de 12 caractere. Mai multe detalii despre codurile folosite la alcătuirea măştilor de introducere a datelor se găsesc în capitolul referitor la Constructorul de forme.

Exemplu

De exemplu, câmpul **Studii** *al tabelei* **PERSONAL** *a fost proiectat de tip şir de caractere cu lungimea 1, pentru memorarea unui caracter cu următoarea semnificaţie:*

- **N** – *nu are studii;*
- **M** – *are studii medii;*
- **I** – *are studii înalte.*

Prin precizarea unei măşti de tipul `A` *se restrânge introducerea în acest câmp doar la litere. Pentru ca ele să fie majuscule, se poate preciza formatul de afişare* `!`. *Cu aceste două îmbunătăţiri, în câmpul* `studii` *nu vor putea fi introduse decât litere, care vor fi transformate automat în majuscule. Rămâne însă problema limitării introducerii în câmp doar la una dintre cele trei litere cu semnificaţie (*`N`, `M` *sau* `I`*), problemă care va fi rezolvată prin introducerea unei clauze de validare la nivel de câmp (acestea sunt tratate într-un paragraf ulterior).*

Valorile implicite ale câmpurilor

O altă facilitate suplimentară a tabelelor legate la o bază de date este cea a valorilor implicite ale câmpurilor. Pentru fiecare câmp al unei astfel de tabele se poate preciza o valoare care să fie încărcată automat la adăugarea unei noi înregistrări.

Evident, valoarea respectivă trebuie să fie de acelaşi tip cu câmpul. În locul valorii se poate preciza o expresie, a cărei evaluare va conduce la valoarea ce se va încărca în câmp.

Exemplu

De exemplu, să presupunem că dorim ca la adăugarea unei noi persoane în tabela `PERSONAL`, *câmpul* `adresa` *să includă implicit oraşul Bucureşti, deoarece majoritatea angajaţilor societăţii comerciale locuiesc în Bucureşti. Pentru aceasta, s-ar putea preciza pentru câmpul* `Adresa` *valoarea implicită* `Bucuresti`, *câmpul fiind automat completat cu acest conţinut ori de câte ori în tabelă este adăugată o nouă înregistrare.*

Un efect mai interesant al acestei facilităţi se obţine prin precizarea ca valoare implicită a câmpului a unei expresii complexe, care poate conţine o funcţie definită de utilizator, în care se pot citi diferite valori, se pot executa diverse operaţii etc.

Exemplu

De exemplu, dacă se doreşte alocarea unui cod numeric pentru fiecare persoană din tabelă, se poate preciza un câmp special, să zicem `Cod`. *Acesta ar avea ca valoare implicită o funcţie, să zicem* `calcul()`, *în care se va calcula maximul valorilor din câmpul respectiv. Funcţia va returna valoarea obţinută plus unu, ceea ce va asigura unicitatea codului respectiv.*

Validarea la nivel de câmp

Pentru un câmp al unei tabele legate poate fi precizată o expresie logică, ce va fi evaluată la modificarea valorii din câmp. Dacă valoarea rezultată în urma evaluării acestei expresii este *fals*, atunci sistemul afişează un mesaj de eroare, fie unul propriu (ca de exemplu „*Regula de validare a câmpului... este violată*"), fie unul specificat de utilizator. Dacă însă valoarea rezultată în urma evaluării expresiei logice este *adevărat*, se consideră că valoarea introdusă în câmp este corectă şi nu se mai generează nici un mesaj de eroare.

Această facilitate ţine de orientarea spre obiecte a sistemului Visual FoxPro şi, mai precis, de modelul programării conduse de evenimente implementat în acesta (secvenţele de cod pentru validare sunt executate numai la apariţia anumitor evenimente, necontrolate direct de programator).

Exemplu

Să luăm spre exemplu câmpul `studii` *al tabelei* `PERSONAL`. *Acest câmp trebuie completat cu unul dintre caracterele* `N`, `M` *sau* `I`. *Într-un paragraf anterior am stabilit pentru acest câmp un format de afişare (care determina trecerea caracterului introdus de utilizator la majusculă) şi o mască de introducere a datelor în câmp, cu ajutorul căreia se limita încărcarea datelor în câmp la litere. Rămăsese de rezolvat problema restrângerii introducerii în câmp numai la cele trei litere, oricare alta fiind invalidă.*

Pentru aceasta se poate folosi validarea la nivel de câmp. Se poate preciza deci pentru câmpul `studii` *o expresie de următoarea formă:*

```
INLIST(Studii, 'N', 'M', 'I')
```

care determină returnarea valorii fals, *dacă se introduce o altă literă decât* `N`, `M` *sau* `I`, *şi* adevărat *în caz contrar.*

Poate fi specificat de asemenea un mesaj de eroare al utilizatorului, mesaj care să fie afişat în locul celui generat automat de sistem.

Exemplu

De exemplu:

```
N (fara studii), M (studii medii) sau I (studii inalte)
```

Obs *Evaluarea condiţiei specificate pentru validarea la nivel de câmp este declanşată la încărcarea unor date în câmp sau la modificarea celor existente.*

Validarea la nivel de câmp se declanşează înaintea celei la nivel de înregistrare şi a secvenţelor de cod asociate evenimentelor de manipulare a tabelei.

Validarea la nivel de înregistrare

Un alt tip de validare a datelor introduse într-o tabelă este cea la nivel de înregistrare. Pentru o tabelă încorporată într-o bază de date se poate preciza o expresie logică ce urmează a fi evaluată automat atunci când se modifică datele dintr-o înregistrare (se adaugă noi date, se modifică sau se şterg unele existente etc.). Această validare nu este declanşată la ştergerea unei întregi înregistrări a tabelei.

Expresia de validare trebuie să returneze un rezultat logic. Dacă acesta este *adevărat*, modificarea efectuată se consideră corectă. Dacă însă rezultatul este *fals*, modificările nu sunt acceptate şi este afişat un mesaj de eroare (fie unul al sistemului, fie unul specificat de utilizator).

În expresia de validare se poate încorpora o funcţie definită de utilizator, în care se pot introduce o multitudine de comenzi şi funcţii de prelucrare complexe.

Exemplu

De exemplu, la modificarea datelor dintr-o tabelă se pot specifica prin intermediul expresiei de validare la nivel de înregistrare o serie de corelaţii între valorile din câmpuri.

Obs

Validarea la nivel de înregistrare se declanşează după cea la nivel de câmp, dar înaintea secvenţelor de cod asociate evenimentelor de manipulare a tabelei.

Secvenţe de cod asociate evenimentelor de manipulare a unei tabele legate

Alte trei evenimente legate de manipularea unei tabele legate sunt cele de adăugare a unei noi înregistrări, de modificare a datelor dintr-o înregistrare existentă şi de ştergere a unei înregistrări a tabelei. Pentru fiecare dintre aceste evenimente se poate specifica o secvenţă de cod care să fie executată la apariţia sa.

Obs

Secvenţele de cod asociate evenimentelor de manipulare a unei tabele legate se declanşează după validările la nivel de câmp şi la nivel de înregistrare.

Caracteristici noi la nivelul bazelor de date în ansamblu

Într-o bază de date de tip nou (container) pot fi incluse mai multe tabele, acestea căpătând astfel o serie de proprietăți noi, ce au fost descrise anterior. Pe lângă caracteristicile noi la nivelul tabelelor componente, o bază de date de tip nou oferă o serie de facilități care nu erau disponibile înainte.

Mai întâi, în cadrul unei astfel de baze de date pot fi stabilite relații permanente între tabelele componente, relații care nu sunt şterse o dată cu părăsirea programelor de prelucrare în care au fost definite (cum este cazul relațiilor temporare între tabele, cele disponibile în vechile versiuni ale SGBD). Datele referitoare la relațiile respective sunt memorate în fişierul bazei de date (.DBC) şi sunt restabilite automat la deschiderea acesteia.

Pentru aceste tipuri de relații sunt disponibile facilități suplimentare, precum integritatea referențială. Deşi are un nume cam pretențios, această caracteristică se reduce în fapt la o serie de reguli privind actualizarea datelor din câmpurile tabelelor legate între ele prin relații permanente.

De exemplu, dacă modificăm un cod dintr-un câmp al unei tabele copil, ce se întâmplă cu codurile corespunzătoare din tabela părinte care indicau spre codul modificat?

În cadrul unei baze de date noi, mai putem include:

- vederi („view" în engleză), care reprezintă tabele virtuale construite pe baza unuia sau mai multor câmpuri din alte tabele;
- proceduri şi funcții asociate bazei de date, care pot fi apelate în secvențele de cod asociate tabelelor componente (de exemplu, în secvența de cod asociată evenimentului de actualizare a unei înregistrări a unei tabele);
- conexiuni, care reprezintă suportul (mecanismul) pentru accesul la datele furnizate de alte sisteme, precum alte sisteme SGBD, programe de calcul tabelar etc.

Aceste proprietăți vor fi prezentate cu ocazia tratării fiecărui subiect în parte, în capitolele următoare.

Relații permanente între tabelele unei baze de date

Modelul relațional al bazelor de date implică organizarea datelor în tabele legate între ele prin relații. Scopul stabilirii relațiilor este acela de coordonare, după diferite criterii, a datelor citite din tabelele aflate în legătură.

Exemplu

De exemplu, pentru tabela **ANGAJATI**, *unul dintre câmpuri este acela al funcției ocupate de respectiva persoană în cadrul unității economice. Memorarea în această tabelă a întregii denumiri a funcției fiecărei persoane nu este avantajoasă, deoarece pot exista mai multe persoane cu aceeaşi funcție. Pentru fiecare dintre acestea se va memora denumirea completă a aceleiaşi funcții, de aici rezultând o risipă de spațiu de memorare.*

Soluția problemei este de a se memora în tabela **ANGAJATI** *doar codul funcției respective, care, evident, este mult mai mic decât întreaga denumire. Separat de tabela* **ANGAJATI** *se va proiecta o nouă tabelă, numită* **FUNCTII**, *în care se va memora fiecare cod de funcție în parte, împreună cu denumirea completă corespunzătoare. Pentru ca selectarea unei persoane din tabela* **ANGAJATI** *să fie însoțită de selectarea funcției corespunzătoare în tabela* **FUNCTII**, *între cele două tabele trebuie stabilită o relație după* **CodFunctie** *(acest câmp existând în ambele tabele).*

Exemplul de mai sus ilustrează o situație în care este necesară stabilirea unei relații între două tabele.

Relațiile dintre două tabele se pot clasifica în următoarele patru clase:

- *una-la-una* – caz în care fiecare înregistrare din tabela părinte este pusă în corespondență cu o înregistrare din tabela copil;
- *una-la-mai-multe* – când unei înregistrări a tabelei părinte îi corespund mai multe înregistrări ale tabelei copil;
- *mai-multe-la-una* – când pentru o înregistrare a tabelei copil putem avea mai multe înregistrări în tabela părinte;
- *mai-multe-la-mai-multe* – caz în care unei înregistrări a tabelei părinte îi corespund mai multe înregistrări ale tabelei copil şi invers, unei înregistrări ale tabelei copil îi corespund mai multe înregistrări ale tabelei părinte.

Dintre acestea, în Visual FoxPro sunt implementate doar primele trei tipuri de relații. Lipsesc relațiile de tip *mai-multe-la-mai-multe*, deoarece ele se pot reduce la două relații, una de tip *mai-multe-la-una* şi alta de tip *una-la-mai-multe* (prin intermediul unei tabele suplimentare intercalate între cele două).

Un alt criteriu de clasificare a relațiilor dintre tabele este cel al momentului definirii lor. Din acest punct de vedere există:

- *relații temporare*, sau *dinamice*, care sunt create prin comenzi introduse în programele de prelucrare, deci sunt disponibile numai la rularea acestora. În

afara programelor de prelucrare, relaţiile respective nu există (de aceea se numesc temporare);

- *relaţii permanente*, care sunt create automat de sistem la deschiderea bazei de date care conţine tabelele legate. Aceste relaţii sunt disponibile imediat ce este deschisă baza de date, deoarece ele sunt memorate în fişierul bazei de date.

Aici ne vom ocupa de relaţiile permanente între tabele. Relaţiile temporare vor fi tratate într-un paragraf special, în capitolul referitor la exploatarea bazelor de date.

La crearea unei relaţii permanente între două tabele este necesară îndeplinirea unor condiţii prealabile şi anume:

- tabela părinte (conducătoare) să fie indexată cu un index candidat sau primar;
- tabela copil (condusă) să fie indexată cu orice fel de index (de acesta din urmă depinde tipul relaţiei stabilite).

Indecşii corespunzători trebuie să existe anterior creării relaţiei.

Condiţia indexării tabelei părinte cu un index candidat sau primar determină stabilirea numai a unor relaţii permanente de tip *una-la-una* sau *una-la-mai-multe*. Aceasta deoarece în tabela părinte după indexul respectiv nu pot să existe două înregistrări cu aceeaşi valoare a cheii de indexare.

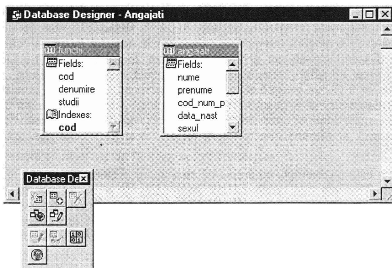
Dacă dorim crearea unei relaţii de tip *mai-multe-la-una*, ea trebuie văzută dinspre tabela copil înspre tabela părinte. Desigur că prin aceasta se inversează rolurile părinte-copil între tabele. Dacă dorim neapărat ca tabela părinte să-şi păstreze acest rol pentru o altă relaţie, se poate crea o vedere din cele două tabele legate, vedere care va deveni noua tabelă părinte pentru relaţia respectivă (această tehnică va fi explicată pe larg în paragraful referitor la vederi).

Să vedem acum modul practic de stabilire a unei relaţii (permanente) între două tabele ale unei baze de date. Dacă fişierul bazei de date a fost creat anterior, este necesară deschiderea sa pentru modificare. Dacă nu, este necesară crearea fişierului respectiv (cu ajutorul opţiunii **New** a submeniului **File** – a se vedea un paragraf anterior).

Deschiderea bazei de date pentru modificare se realizează prin alegerea opţiunii **Open** (deschidere) din submeniul **File** (fişiere). După specificarea bazei de date respective într-o fereastră de dialog este pornit Constructorul de baze de date.

Urmează apoi adăugarea la baza de date a celor două tabele, dacă acest lucru nu s-a realizat anterior. Operaţia se realizează cu ajutorul opţiunii **Add table** (adăugare tabelă) din meniul **Database** (bază de date).

După realizarea acestor operaţii, fereastra Constructorului de baze de date ar putea arăta astfel:



Presupunând că au fost deja creaţi indecşii necesari stabilirii relaţiei, urmează acum stabilirea relaţiei permanente între cele două tabele. Pentru aceasta se execută un clic simplu pe indexul (de tip candidat sau primar) care dă expresia relaţiei din tabela părinte şi se trage cu mouse-ul peste indexul (de orice tip) care dă expresia relaţiei din tabela copil.

De exemplu, în cazul tabelei FUNCTII legate ca părinte de tabela ANGAJATI, vom trage cu mouse-ul indexul `cod` (din prima dintre tabelele amintite) peste indexul `functie` (al celei de-a doua tabele). Tipul indexului `cod` al tabelei FUNCTII dă şi tipul relaţiei ce se stabileşte între cele două tabele. Dacă indexul este de tip normal sau unic, atunci se va stabili o relaţie de tip *una-la-una*. Dacă indexul este de tip candidat sau cheie primară, relaţia va fi de tip *una-la-mai-multe*.

Integritatea referenţială sau regulile de actualizare a tabelelor legate între ele prin relaţii

Integritatea referenţială reprezintă un ansamblu de reguli impuse tabelelor între care s-au stabilit relaţii. Aceste reguli sunt necesare deoarece deseori se doreşte modificarea unor date dintr-o tabelă, date care afectează relaţia dintre această tabelă şi o alta.

Să luăm, de exemplu, relaţia stabilită între tabelele FUNCTII şi ANGAJATI. Aceasta este o relaţie de tip *una-la-mai-multe*. Tabela părinte este FUNCTII, iar tabela copil este ANGAJATI. Expresia după care s-a stabilit legătura între aceste două tabele este un

câmp existent în ambele tabele, codul funcției (chiar dacă numele diferă în cele două tabele). Să presupunem următoarea situație: în tabela ANGAJATI există doi directori, iar în tabela FUNCTII o singură înregistrare referitoare la această funcție. În cazul în care unul dintre directori pleacă din unitatea respectivă, înregistrarea lui trebuie ştearsă. Dacă ar fi existat un singur director în întreaga tabelă a angajaților, atunci, o dată cu ştergerea sa, ar fi trebuit ştearsă şi înregistrarea corespunzătoare din tabela FUNCTII, deoarece aceasta nu mai foloseşte la nimic. Dacă însă se şterge un singur director din doi sau mai mulți, înregistrarea din tabela FUNCTII nu trebuie ştearsă, deoarece în tabela ANGAJATI ar rămâne directori care nu ar mai avea funcția descrisă în tabela funcțiilor.

Acesta este un exemplu de problemă care apare la ştergerea unor înregistrări din tabele legate între ele prin relații. Situații asemănătoare ar putea apărea şi în cazul modificării datelor. De exemplu, ce se va întâmpla dacă se doreşte schimbarea codului funcției de director? Nu este de ajuns modificarea acestui cod în înregistrarea din tabela FUNCTII, deoarece înregistrările corespunzătoare din tabela ANGAJATI ar rămâne cu vechile coduri, deci ar indica funcții inexistente.

Integritatea referențială este specifică unei relații dintre două tabele. Ea se aplică doar în cazul modificării acelor date dintr-o înregistrare care afectează relația respectivă. Printre evenimentele care conduc la modificări ale cheii şi deci reprezintă evenimente tratate prin integritatea referențială, se numără:

- adăugarea unei înregistrări noi;
- ştergerea unei înregistrări;
- modificarea datelor unei înregistrări, date care afectează relația.

În cazul adăugării unei noi înregistrări în tabela copil a unei relații, există următoarele opțiuni:

- **ignorare** (*Ignore*) – cu alte cuvinte, se permite adăugarea noii înregistrări în tabela copil, indiferent dacă există sau nu o înregistrare corespunzătoare în tabela părinte;
- **restricționare** (*Restrict*) – se generează o eroare atunci când se încearcă adăugarea unei înregistrări în tabela copil, fără corespondent în tabela părinte a relației.

Exemplu

Acest caz apare, de exemplu, la încercarea de adăugare a unei persoane în tabela ANGAJATI, fără ca anterior să se fi introdus în tabela FUNCTII o înregistrare cu datele referitoare la funcția ocupată de aceasta.

Ştergerea unei înregistrări din tabela părinte a unei relaţii este de asemenea gestionată prin intermediul integrităţii referenţiale. În acest caz, avem la dispoziţie următoarele opţiuni:

- **ignorare** (*Ignore*) – se permite această ştergere, indiferent dacă în tabela copil există sau nu înregistrări legate de înregistrarea părinte ştearsă;
- **restricţionare** (*Restrict*) – opreşte ştergerea, generând un mesaj de eroare, atunci când există înregistrări corespunzătoare în tabela copil;
- **ştergere în cascadă** (*Cascade*) – se şterg automat toate înregistrările din tabela copil legate de înregistrarea părinte ştearsă.

Exemplu

De exemplu, la ştergerea înregistrării antet a unei facturi, trebuie şterse toate înregistrările cu articole din tabela copil (cazul „ştergere în cascadă").

Modificarea unor date din tabela părinte, modificare care afectează relaţia dintre tabele, este de asemenea tratată prin regulile integrităţii referenţiale. În acest caz, avem la dispoziţie următoarele variante:

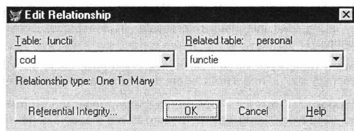
- **ignorare** (*Ignore*) – se permit modificările respective, chiar dacă prin aceasta înregistrările copil corespunzătoare rămân „în aer", adică fără corespondent în tabela părinte;
- **restricţionare** (*Restrict*) – atunci când există înregistrări corespunzătoare în tabela copil, modificarea este oprită şi este generat un mesaj de eroare;
- **modificare în cascadă** (*Cascade*) – sunt modificate automat toate înregistrările din tabela copil conform noii valori a cheii relaţiei.

Exemplu

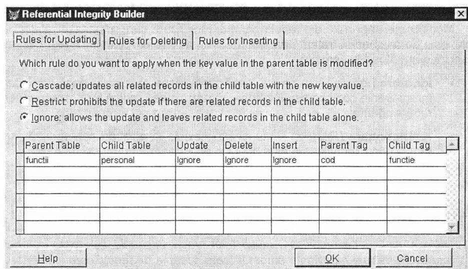
De exemplu, dacă modificăm codul unei funcţii în tabela FUNCTII, în tabela ANGAJATI trebuie modificate automat toate codurile de funcţii corespunzătoare („modificare în cascadă"), pentru a păstra aceleaşi funcţii pentru respectivele persoane.

Specificarea opţiunilor legate de integritatea referenţială a unei relaţii se face astfel:

- în fereastra Constructorului de baze de date se execută dublu clic pe relaţia respectivă (pe linia frântă care leagă tabelele). Se deschide pe ecran fereastra de dialog **Edit Relationship** (editare relaţie), în care se precizează parametrii relaţiei respective;



- în această fereastră se apasă butonul **Referential Integrity** (integritate referențială), care determină deschiderea ferestrei cu același nume, în care se vor specifica opțiunile legate de acest subiect;



- cele trei pagini ale ferestrei controlează fiecare câte un tip de eveniment, astfel:
    - **Rules for Updating** (reguli pentru actualizare) – pagina cu opțiuni referitoare la modificarea datelor din tabelele legate;
    - **Rules for Deleting** (reguli pentru ștergere) – pagina cu opțiuni referitoare la ștergerea înregistrărilor părinte ale unei relații;

- **Rules for Inserting** (reguli pentru inserare) – pagina cu opţiuni referitoare la adăugarea de noi înregistrări la tabelele legate.

Vederi

Def

***Vederile** reprezintă un tip special de tabele, construite pe baza datelor din una sau mai multe tabele sau vederi, legate între ele prin relaţii.*

Vederile sunt folosite atunci când se doreşte o altă structură a tabelelor (bazelor de date), structură construită pe baza unor tabele şi vederi deja existente. Vederile sunt incluse în bazele de date, ele neputând fi folosite decât atunci când baza de date care le conţine este deschisă. Chiar memorarea efectivă a vederilor se face în fişierul bazei de date (.DBC). Vederile sunt văzute şi prelucrate ca orice altă tabelă.

Exemplu

De exemplu, pentru baza de date a angajaţilor unităţii economice se doreşte o tabelă cu numele, prenumele şi funcţia (denumirea completă) fiecărei persoane din baza de date. Deşi datele necesare au fost introduse anterior în tabelele ANGAJATI şi FUNCTII, nu există o tabelă care să conţină exact câmpurile amintite. Numele şi prenumele se găsesc în tabela ANGAJATI, iar denumirea funcţiilor în tabela FUNCTII. Aceste două tabele sunt legate între ele (în cadrul bazei de date) prin câmpul codului funcţiei.

Pentru problema enunţată mai sus se poate construi o vedere, care să fie şi ea inclusă în baza de date şi folosită ori de câte ori se doreşte furnizarea datelor în formatul respectiv.

Obs

*Vederile reprezintă de fapt un alt „punct de **vedere**" asupra datelor dintr-un ansamblu de tabele (baze de date). Ele introduc un nou nivel, logic, între tabele şi utilizator. Acesta, având structura fizică a tabelelor, poate obţine orice structură logică doreşte prin intermediul unei vederi.*

O dată construită o vedere, între tabelele sursă şi aceasta se stabileşte o legătură care conduce la actualizarea datelor din vedere atunci când se modifică datele din tabelele respective.

În schimb, actualizarea tabelelor sursă cu modificările efectuate în vedere se realizează în funcţie de o serie de parametri şi numai atunci când este posibil. Aceşti parametri sunt precizaţi la construirea vederii, în Constructorul de vederi.

Există situaţii când actualizarea datelor din tabelele sursă cu modificările din vedere nu este posibilă. De exemplu, o vedere poate conţine câmpuri calculate. Aceste

câmpuri reprezintă de fapt expresii în care intră câmpurile tabelelor sursă. Modificând un astfel de câmp, nu există posibilitatea de determinare a valorilor corespunzătoare din câmpurile sursă.

Exemplu

De exemplu, dacă într-un câmp calculat prin suma a două câmpuri sursă se introduce valoarea 6, aceasta ar putea fi obţinută din 0+6, 1+5 etc., deci nu se pot determina unic valorile din câmpurile sursă respective.

Un exemplu de utilizare a vederilor este acela al relaţiilor de tip *mai-multe-la-una*. Deoarece sistemul nu permite definirea directă a acestor relaţii, ele pot fi simulate cu ajutorul relaţiilor de tip *una-la-mai-multe* şi al unei vederi.

Exemplu

Să luăm exemplul tabelelor ANGAJATI *şi* FUNCTII. *Aceste două tabele trebuie legate printr-o relaţie, astfel încât oricând selectăm în tabela* ANGAJATI *o anumită persoană, în tabela* FUNCTII *să avem funcţia corespunzătoare. Aceasta este o relaţie de tip* mai-multe-la-una, *deoarece există în tabela* ANGAJATI *mai multe persoane care au aceeaşi funcţie (de exemplu, există atâţia şefi de ateliere câte ateliere are unitatea).*

Legând cele două tabele printr-o relaţie de tip una-la-mai-multe, în care părintele este tabela FUNCTII *şi copilul tabela* ANGAJATI, *se obţine efectul dorit. Numai că schimbarea celor două roluri are alte efecte. De exemplu, dacă dorim să parcurgem toţi angajaţii în ordinea vechimii în unitate, nu putem să facem acest lucru deoarece tabela conducătoare este* FUNCTII. *Putem însă să construim o vedere pe baza acestor două tabele, vedere în care să introducem câmpurile care ne interesează (numele, prenumele, denumirea completă a funcţiei, vechimea în muncă etc.). Această vedere o vom ordona după vechimea în muncă, pentru a obţine situaţia solicitată. Mai mult chiar, vederea o putem folosi la construirea altei relaţii, precum cea cu tabela* COMPART, *în care sunt descrise compartimentele în care lucrează fiecare angajat.*

Construirea unei vederi implică parcurgerea următoarelor etape principale:

- dacă baza de date care urmează să conţină vederea nu este deschisă, se deschide, eventual se creează;
- apoi se porneşte Constructorul de vederi;
- urmează precizarea sursei de date, adică a tabelelor din care se preiau datele pentru construirea vederii respective;
- apoi se stabilesc câmpurile care vor intra în componenţa vederii, câmpuri selectate dintre cele ale tabelelor sursă precizate anterior;

- în continuare, se specifică o serie de parametri precum: legăturile între tabelele specificate, eventualele condiţii de filtrare (selecţie), ordonare, grupare a datelor în vedere, modul de actualizare a datelor sursă pe baza datelor din vedere etc.

Pentru construirea vederilor se foloseşte un utilitar special al SGBD-ului Visual FoxPro numit Constructorul de vederi. Modul de utilizare a acestui utilitar este asemănător celui folosit pentru construirea interogărilor, prezentat în detaliu în capitolul referitor la interogarea bazelor de date.

Vederile sunt folosite şi pentru accesul la date de alte tipuri decât tabelele şi bazele de date Visual FoxPro şi la date situate la distanţă (pe servere ale reţelei). **Vederile la distanţă** sunt prezentate în paragraful referitor la tehnologia client/server implementată în Visual FoxPro.

Elemente de proiectare a bazelor de date relaţionale. Tehnica normalizării tabelelor

Proiectarea unei tabele implică specificarea structurii sale (a câmpurilor componente şi a caracteristicilor acestora), dar şi a cheilor folosite pentru identificarea datelor înregistrărilor). În cele ce urmează ne vom ocupa de împărţirea datelor unei baze de date complexe în mai multe tabele simple şi stabilirea legăturilor între acestea problema proiectării tabelelor simple fiind considerată cunoscută).

O metodologie riguroasă de proiectare a unei baze de date relaţionale prevede mai întâi construirea unei structuri iniţiale a bazei de date, urmată de ameliorarea acestei structuri până la obţinerea unei variante optime, normalizate. Varianta de plecare a bazei de date conţine, în general, o serie de anomalii, care sunt corectate în etapa a doua a proiectării, cea de ameliorare sau „normalizare" a bazei de date.

Def ***Normalizarea*** *unei baze de date reprezintă procesul de transformare succesivă a unei baze de date relaţionale în vederea aducerii sale la o formă standard optimizată.*

Normalizarea se realizează în trepte. La fiecare pas, baza de date este trecută într-o nouă formă standard, numită **formă normală**. Procesul de aducere la o anumită formă normală constă, de fapt, în eliminarea unei anumite anomalii, a unei anumite dependenţe nedorite între date, a unei anumite redundanţe. Normalizarea unei baze de date nu este o condiţie obligatorie pentru lucrul cu date organizate conform modelului relaţional. Regulile ce vor fi prezentate în continuare nu trebuie luate ca „legi" ce nu pot fi încălcate niciodată. Există chiar cazuri când este mai potrivită o structură neoptimizată conform acestor reguli.

Nu vom aborda aici în detaliu metodologia de normalizare a unei baze de date relaţionale („tehnica normalizării tabelelor"), ci o vom prezenta doar la nivel principial. Vom folosi pentru exemplificare o bază de date în care se doreşte memorarea facturilor emise şi primite de o unitate economică.

Un exemplu de factură este prezentat în figura de mai jos.

Furnizor		Cumpărător ....
....		....
(den.,forma jurid.)	FACTURA	(den.,forma jurid.) ....
Nr.ord.registru com.		Nr.ord.registru com. ....
Nr.înreg.fiscală	Seria FE Nr.409876	Nr.înreg.fiscală ....
Localitatea		Localitatea ....
Judeţul		Judeţul ....
Contul		Contul ....
Banca		Banca ....

Nr.factură ....
Data (zi,lună,an) ....
Nr.aviz însoţire marfă ......

Nr. crt.	Denumirea produselor sau serviciilor	U.M.	Cantitatea	Preţ unitar (fără TVA) lei	Valoarea lei	Valoarea TVA lei
0	1	2	3	4	5	6

Semnătura şi ştampila Furnizorului	Date privind expediţia Numele delegatului .... mijlocul de transport .... nr. .... Expedierea s-a făcut la data de .... ora .... Semnăturile..................	Total	
		Semnătura de primire	Total de plată (col.5+6)

O factură însoţeşte actul de vânzare-cumpărare a unuia sau mai multor articole (produse sau servicii). Ea este compusă dintr-un antet, care conţine date referitoare la actul de vânzare în ansamblul său (date despre vânzător şi cumpărător, data calendaristică la care se face vânzarea, numărul de înregistrare al facturii etc), şi dintr-o secţiune în care sunt descrise articolele (produsele) ce fac obiectul vânzării (denumirea, cantitatea şi preţul unitar, TVA etc). Numărul de articole vândute printr-o singură factură diferă de la caz la caz, adică este variabil de la o factură la alta.

Vom propune pentru început o structură a bazei de date FACTURI formată dintr-o singură tabelă, de următoarea formă:

FACTURI						
Număr factură	**Data**	**Seria**	**Vânzător (denumire, cod fiscal, cont, banca)**	**Cumpărător (denumire, cod fiscal, cont, banca)**	**Mijloc de transport**	...
...	...	...	...	...	...	...

Observăm că fiecare factură este identificată printr-un cod numeric (numărul facturii), care, pentru a putea fi folosit pe post de cheie primară a tabelei, trebuie să fie unic în cadrul bazei de date – câmpul a fost subliniat, pentru a pune în evidenţă această funcţie a sa. În structura de mai sus nu a fost detaliată partea referitoare la articolele facturii.

Aducerea bazei de date la prima formă normală (FN1)

Pentru a trece o bază de date la prima formă normală – FN1 – trebuie eliminate câmpurile **compuse** (sau neatomice) şi cele **repetitive**. Primul tip de câmpuri amintit (cele compuse) este format din acele câmpuri care reprezintă o concatenare a mai multor valori. În exemplul de mai sus, câmpul Vânzător (la fel şi Cumpărător) reprezintă de fapt o concatenare a multor date simple: codul vânzătorului, denumirea, codul său fiscal, contul şi banca.

Câmpul Vânzător, fiind unul compus, se va „sparge" în mai multe câmpuri elementare, după cum se vede în figura de mai jos:

FACTURI									
Număr factură	**Data**	**Seria**	**Cod vânzător**	**Denumire vânzător**	**Cod fiscal vânzător**	**Cont vânzător**	**Bancă vânzător**	**Cod cumpărător**	...
...	...	...	...	...	...	...	...	...	...

Cel de-al doilea tip de câmpuri care trebuie prelucrate în această etapă (de trecere la prima formă normală) este cel al câmpurilor repetitive. Acestea descriu acelaşi tip de entităţi, numărul lor fiind însă variabil.

Pentru a pune în evidenţă câmpurile repetitive, să detaliem structura tabelei `FACTURI`, în secţiunea referitoare la articolele facturilor.

FACTURI										
<u>Număr factură</u>	...	Articol 1					Articol 2		Articol 3	
		Cod	Denumire	Unitate de măsură	Cantitate	...	Cod	...	Cod	...
...	...	...	...	...	...	...	...		...	...

Observăm în structura de mai sus că într-o înregistrare a tabelei `FACTURI` (în care este memorată complet o singură factură) apar date referitoare la trei articole. Prin urmare, o factură memorată în această bază de date poate conţine maximum trei articole, ceea ce reprezintă pentru utilizator o serioasă limitare. Am putea mări acest număr la, să zicem, 20. Acum, numărul de articole poate satisface un contabil nepretenţios, dar ce se întâmplă dacă majoritatea facturilor conţin 1 sau 2 articole? În acest caz, majoritatea înregistrărilor bazei de date vor conţine 18 sau 19 articole necompletate, ceea ce constituie o mare risipă de spaţiu de memorare.

Câmpurile `Articol 1`, `Articol 2` şi `Articol 3`, puse în evidenţă în exemplul de mai sus, reprezintă câmpuri repetitive. Ele trebuie eliminate din baza de date, operaţie care se face astfel: se introduce în baza de date un singur asemenea câmp, o factură urmând a se întinde acum pe mai multe înregistrări ale tabelei (atâtea înregistrări câte articole are factura). Pentru a ţine evidenţa articolelor din cadrul unei facturi, vom introduce în tabelă un nou câmp, pe care îl vom denumi `Număr articol`.

Noua structură a bazei de date `FACTURI` va fi următoarea:

FACTURI									
<u>Număr factură</u>	Data	Seria	Cod vânzător	Denumire vânzător	...	<u>Număr articol</u>	Cod articol	Denumire articol	...
...	...	...	...	...	...	...	...	...	...

Observăm în structura de mai sus că, pentru identificarea unei înregistrări din tabelă, se folosesc două câmpuri: numărul facturii şi numărul articolului din cadrul acesteia. Prin urmare, noua cheie a tabelei va fi construită prin concatenarea celor două câmpuri (subliniate în structura de mai sus).

Aducerea bazei de date la a doua formă normală (FN2)

O dată baza de date adusă la prima formă normală, se trece la următoarea etapă de optimizare (normalizare), cea de aducere la a două formă normală – FN2. Această a doua formă a bazei de date se caracterizează prin faptul că nu conține **dependențe funcționale parțiale**, ceea ce înseamnă că toate câmpurile tabelei depind doar de cheia primară a acesteia (nu de părți ale acesteia).

O dată operate schimbările anterioare, în tabela facturi apar o serie de anomalii, ce trebuie eliminate. Să observăm câmpul `Data` (și alte câmpuri din antetul facturii), care nu depind biunivoc de noua cheie primară a tabelei, formată din câmpurile `Număr factură` și `Număr articol`, ci doar de o parte a acesteia, adică de câmpul `Număr factură`. Acesta este un exemplu de dependență funcțională parțială ce trebuie eliminată din tabelă, pentru ca ea să fie de forma FN2.

Soluția este mutarea câmpurilor care formează antetul facturii (care depind doar de `Număr factură`) într-o nouă tabelă, împreună cu câmpul de care depind (adică `Număr factură`). Noile tabele le vom numi `ANTET` și `ARTICOLE`, păstrând numele de `FACTURI` pentru întreaga bază de date.

ANTET				
Număr factură	Data	Seria	Cod vânzător	...
...	...	...	...	...

ARTICOLE			
Număr factură	Număr articol	Cod articol	...
...	...	...	...

Fiecare factură va fi memorată în baza de date (care acum este compusă din două tabele) printr-o singură înregistrare în tabela `ANTET` și una sau mai multe înregistrări în tabela `ARTICOLE` (atâtea înregistrări câte articole conține factura). Legătura între cele două tabele este realizată printr-un câmp comun, `Număr factură`.

Observăm, de asemenea, cheile celor două tabele: `Număr factură` în tabela `ANTET` și combinația `Număr factură – Număr articol` în tabela `ARTICOLE`.

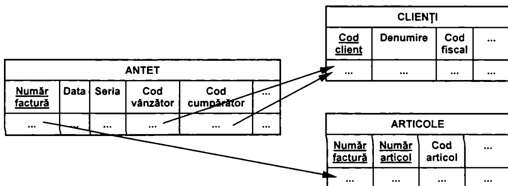
Aducerea bazei de date la a treia formă normală (FN3)

A treia formă normală a unei baze de date se caracterizează prin aceea că nu conține **dependențe funcționale tranzitive**. Aceste dependențe se obțin atunci când un câmp X depinde de un alt câmp Y, care, la rândul lui, depinde de un altul Z. În același timp, X depinde și direct de Z. Aceste dependențe din cadrul unei tabele trebuie eliminate, în caz contrar putând apărea o serie de anomalii nedorite.

Să luăm, de exemplu, câmpul Denumire vânzător, care depinde biunivoc de câmpul Cod vânzător (fiecare vânzător, identificat printr-un cod unic, are o denumire unică). La rândul lui, câmpul Cod vânzător depinde biunivoc de câmpul cheie primară a tabelei ANTET, adică de câmpul Număr factură. Dar câmpul Denumire vânzător depinde şi direct de câmpul Număr factură, deoarece fiecare factură are o singură denumire de vânzator.

Eliminarea acestei dependenţe se realizează prin construirea unei noi tabele, folosite la memorarea datelor despre vânzătorii înscrişi pe facturi. Dar cum aceeaşi situaţie se întâlneşte şi în cazul datelor despre cumpărători, vom construi o tabelă mai generală, în care vom introduce toţi vânzătorii şi cumpărătorii, tabelă pe care o vom denumi CLIENŢI. În această tabelă vom introduce toate câmpurile conţinând date despre clienţi, împreună cu codul acestora şi cu cheia primară a tabelei din care au fost extrase câmpurile (adică Număr factură din tabela ANTET). În tabela ANTET va rămâne câmpul Cod vânzător (şi Cod cumpărător), pentru a face astfel legătura între cele două tabele.

Prin urmare, vom avea următoarea structură a tabelei:

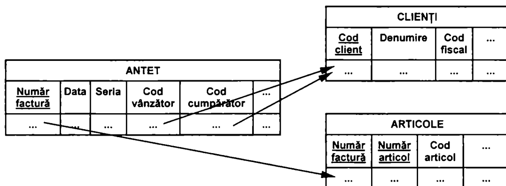


Aceeaşi problemă apare şi în cazul produselor din componenţa articolelor, care sunt caracterizate de Cod, Denumire şi Unitate de măsură. Câmpul Denumire (ca şi Unitate de măsură) depinde de Cod Articol, care, la rândul lui, depinde de cheia primară a tabelei ARTICOLE (combinaţia dintre Număr factură şi Număr articol). Prin urmare, vom construi o nouă tabelă, numită PRODUSE, cu următoarea structură:

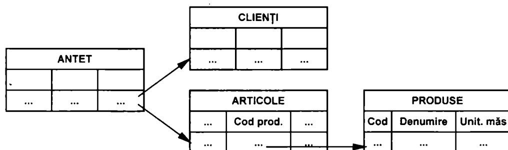
Să luăm, de exemplu, câmpul `Denumire vânzător`, care depinde biunivoc de câmpul `Cod vânzător` (fiecare vânzător, identificat printr-un cod unic, are o denumire unică). La rândul lui, câmpul `Cod vânzător` depinde biunivoc de câmpul cheie primară a tabelei `ANTET`, adică de câmpul `Număr factură`. Dar câmpul `Denumire vânzător` depinde şi direct de câmpul `Număr factură`, deoarece fiecare factură are o singură denumire de vânzator.

Eliminarea acestei dependenţe se realizează prin construirea unei noi tabele, folosite la memorarea datelor despre vânzătorii înscrişi pe facturi. Dar cum aceeaşi situaţie se întâlneşte şi în cazul datelor despre cumpărători, vom construi o tabelă mai generală, în care vom introduce toţi vânzătorii şi cumpărătorii, tabelă pe care o vom denumi `CLIENŢI`. În această tabelă vom introduce toate câmpurile conţinând date despre clienţi, împreună cu codul acestora şi cu cheia primară a tabelei din care au fost extrase câmpurile (adică `Număr factură` din tabela `ANTET`). În tabela `ANTET` va rămâne câmpul `Cod vânzător` (şi `Cod Cumpărător`), pentru a face astfel legătura între cele două tabele.

Prin urmare, vom avea următoarea structură a tabelei:



Aceeaşi problemă apare şi în cazul produselor din componenţa articolelor, care sunt caracterizate de `Cod`, `Denumire` şi `Unitate de măsură`. Câmpul `Denumire` (ca şi `Unitate de măsură`) depinde de `Cod Articol`, care, la rândul lui, depinde de cheia primară a tabelei `ARTICOLE` (combinaţia dintre `Număr factură` şi `Număr articol`). Prin urmare, vom construi o nouă tabelă, numită `PRODUSE`, cu următoarea structură:



Formele normale 4 şi 5

În continuare, baza de date în FN3 mai poate fi optimizată, prin aducerea la formele FN4 şi FN5. Pentru aducerea unei tabele în forma FN4 este necesară eliminarea **dependențelor multivaloare** suplimentare (pentru o tabelă este permisă numai una singură), iar FN5 implică lipsa **dependențelor joncțiune**. Nu vom intra aici în detalii legate de aceste forme normale.

Conchizând, am stabilit următoarea structură a bazei de date FACTURI:

tabela ANTET:

```
Număr factură
Data
Seria
Cod vânzător
Cod cumpărător
...
```

tabela ARTICOLE:

```
Număr factură
Număr înregistrare
Cod produs
Cantitatea
...
```

tabela CLIENȚI:

```
Cod client
Denumirea
Cod fiscal
Cont
Banca
Adresa
...
```

tabela PRODUSE:

```
Cod produs
Denumire
Unitate de măsură
```

Exploatarea bazelor de date

Capitolul 4

- Modul de lucru cu bazele de date relaţionale
- Prelucrarea interactivă a datelor din bazele de date
  - ✓ Deschiderea bazelor de date şi a tabelelor
  - ✓ Editarea conţinutului tabelelor. Fereastra **Browse**
- Folosirea limbajului Visual FoxPro pentru prelucrarea datelor din bazele de date
  - ✓ Deschiderea şi închiderea tabelelor şi a bazelor de date
  - ✓ Indicatorul de înregistrări. Înregistrarea curentă
  - ✓ Prelucrarea grupurilor de înregistrări. Domeniul înregistrărilor
  - ✓ Adăugarea de înregistrări la o tabelă
  - ✓ Afişarea conţinutului unei tabele
  - ✓ Modificarea conţinutului unei tabele
  - ✓ Ştergerea înregistrărilor dintr-o tabelă
  - ✓ Accesul la înregistrările tabelei. Filtrarea
  - ✓ Căutarea datelor în tabele
  - ✓ Realizarea de calcule statistice cu datele din tabele
  - ✓ Ordonarea datelor din tabele
  - ✓ Tipuri speciale de câmpuri. Câmpurile „memo" şi câmpurile „generale"
  - ✓ Baze de date relaţionale. Relaţii între tabele

Modul de lucru cu bazele de date relaţionale

Bazele de date reprezintă structuri complexe folosite la gestionarea datelor descriind un anumit tip de entităţi. Conform modelului relaţional, o bază de date reprezintă un ansamblu de tabele, legături între acestea, alte tipuri de fişiere şi alte elemente, folosite împreună pentru memorarea datelor.

La nivelul sistemului de operare, bazele de date şi elementele componente ale acestora sunt memorate în fişiere. Modul clasic de lucru cu fişierele este următorul: mai întâi acestea se creează. O dată creat, ori de câte ori se doreşte prelucrarea unui fişier, el trebuie deschis. După aceea se pot executa o serie de operaţii cu datele pe care le conţine. În final, fişierul trebuie închis.

Acest mod de lucru este specific şi tabelelor şi bazelor de date. Prin urmare, crearea unei tabele şi a unei baze de date trebuie să preceadă folosirea acestora. Pentru a opera cu datele dintr-o tabelă deja creată, ea trebuie mai întâi deschisă. La deschidere, sistemul îi alocă o zonă de memorie specială, numită **zonă de lucru**. În această zonă de memorie sunt memoraţi o serie de parametri ai tabelei, parametri necesari gestiunii tabelei, ca de exemplu numărul înregistrării curente, numărul total de înregistrări din tabelă etc.

Într-o zonă de lucru nu poate fi deschisă decât o singură tabelă. Lucrul cu două sau mai multe tabele simultan implică folosirea a două sau mai multe zone de lucru distincte. Visual FoxPro posedă o multitudine de zone de lucru (de ordinul zecilor de mii), astfel încât, practic, nu există restricţii din acest punct de vedere.

După deschidere, asupra tabelei se pot executa o serie de operaţii de prelucrare, dintre care cele mai importante sunt: adăugarea de noi înregistrări, modificarea unor date din înregistrările existente, ştergerea unor înregistrări, căutarea anumitor date, ordonarea datelor după diferite criterii etc. În acest capitol vor fi prezentate cele mai multe dintre aceste operaţii. Când se termină lucrul cu o tabelă, aceasta trebuie închisă, eliberându-se astfel zona de lucru respectivă.

O observaţie importantă legată de lucrul cu datele din tabele este că operaţiile se efectuează la nivel de înregistrare. Prin urmare, într-o tabelă nu putem adăuga sau şterge decât o înregistrare completă. Pentru a ţine evidenţa înregistrării curent prelucrate dintr-o tabelă, sistemul foloseşte o variabilă specială, numită **indicatorul de înregistrări**. Această variabilă indică totdeauna înregistrarea asupra căreia va acţiona următoarea comandă. Veţi afla mai multe despre această variabilă în paragraful special destinat acestui subiect în capitolul de faţă.

Bazele de date reprezintă, de asemenea, fişiere pe discurile sistemului de calcul. Prin urmare, şi ele trebuie deschise înainte de folosire şi închise după aceea. Ori de câte ori avem nevoie de o informaţie memorată în fişierul bazei de date, este necesară deschiderea bazei de date. Acesta este cazul caracteristicilor suplimentare ale tabelelor legate (nume lungi pentru tabelă şi câmpuri, secvenţe de cod ataşate tabelei etc.), care, pentru a fi disponibile, presupun în prealabil deschiderea bazei de date.

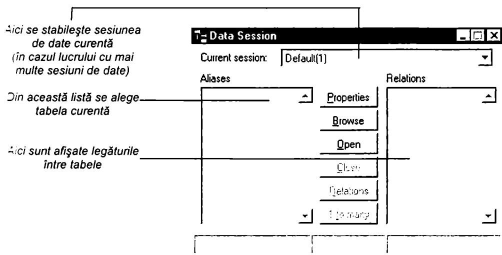
Pentru prelucrarea datelor din bazele de date se pot folosi două metode:

- una *interactivă*, constând din utilizarea diferitelor instrumente ale mediului Visual FoxPro. Operaţiile de prelucrare sunt realizate prin intermediul interfeţei sistemului, sarcina utilizatorului constând în alegerea de opţiuni, întrebuinţarea unor ferestre de dialog etc. Metoda este folosită atunci când operaţiile de prelucrare sunt executate de un cunoscător al mediului şi nu în mod repetat (caz în care este mai avantajoasă realizarea unui program, a unei forme, care să automatizeze operaţiile respective);
- una *prin cod*, sau prin intermediul limbajului FoxPro. Această metodă constă în folosirea diferitelor comenzi ale limbajului, care sunt introduse fie direct în fereastra de comenzi, fie în diferite programe sau metode ale unor forme, rapoarte etc. Realizarea unui sistem informatic necesită folosirea acestei metode, prin construirea unor forme, rapoarte şi a altor elemente, în ale căror metode sunt introduse comenzi de prelucrare a datelor din bazele de date.

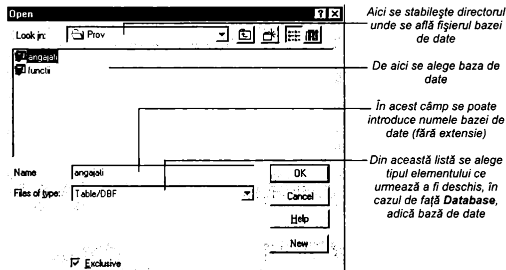
Prelucrarea interactivă a datelor din bazele de date

Deschiderea bazelor de date şi a tabelelor

Înainte de prelucrarea datelor dintr-o bază de date, aceasta trebuie deschisă; la fel şi tabelele componente. Pentru realizarea acestor operaţii vom folosi fereastra **Data Session** (sesiunea de date), deschisă la alegerea opţiunii cu acelaşi nume a meniului **Window**.



Pentru deschiderea unei tabele se acţionează butonul **Open** (deschidere) din această fereastră, care deschide pe ecran o fereastră de dialog pentru selectarea tabelei dorite.



O dată deschisă, tabela va apărea în lista din stânga a ferestrei **Data Session**. Dacă tabela este una legată (încorporată într-o bază de date), deschiderea tabelei implică deschiderea automată a fişierului bazei de date respective.

Editarea conţinutului tabelelor. Fereastra Browse

Principalul instrument pentru realizarea diferitelor operaţii de prelucrare asupra datelor din tabele este fereastra **Browse**. Aici este afişat conţinutul tabelei (în mod implicit într-un format tabelar, coloanele reprezentând câmpuri, iar liniile înregistrări), conţinut care poate fi modificat de utilizator în mod interactiv. Se pot adăuga aici noi înregistrări sau şterge înregistrări existente, se poate interveni asupra conţinutului oricărui câmp şi al oricărei înregistrări, se pot căuta diferite date etc.

Deschiderea unei ferestre **Browse** pentru o tabelă se realizează prin selectarea tabelei respective în fereastra **Data Session** (tabela trebuie să fi fost deschisă anterior), urmată de acţionarea butonului **Browse.**

Angajati

Marca	Nume	Prenume	Functia
101	Popa	Alexandru	Director gener
102	Toma	Dumitru	Director execu
103	Popescu	Ioana Maria	Contabil sef
104	Constantinescu	Viorica	Consilier 2
105	Vasiliu	Rodica	Secretara
106	Dimoftache	Constantin	Referent 3
107	Musat	Florica Violeta	Secretara
108	Petrina	Luminita	Inginer sef
109	Elefterescu	Carmen	Referent 2

O dată cu deschiderea ferestrei **Browse**, la meniul sistemului este adăugat un nou submeniu, numit **Table**, care conține comenzile pentru realizarea diferitelor prelucrări asupra tabelei.

Properties
Font...

Go to Record ▸
Append New Record Ctrl+Y
Toggle Deletion Mark Ctrl+T

Append Records...
Delete Records...
Recall Records...
Remove Deleted Records

Replace Field...
Size Field
Move Field

Resize Partitions
✓ Link Partitions
Change Partitions Ctrl+H

Adăugarea de noi înregistrări

Adăugarea unei noi înregistrări la tabela deschisă spre editare într-o fereastră **Browse** se realizează prin alegerea opțiunii **Append New Record** (adăugare înregistrare nouă). Opțiunea are asociată şi o cale directă de selectare, Ctrl+Y, la a cărei acționare este executată direct operația de adăugare. Ca urmare a oricăreia dintre

metodele de mai sus, în fereastra **Browse** apare pe ultima poziţie o nouă înregistrare goală, care, ulterior, poate fi încărcată cu datele dorite.

Modificarea datelor

Modificarea datelor din oricare înregistrare a tabelei se poate face direct în fereastra **Browse**, prin deplasarea cursorului în câmpul şi înregistrarea respectivă şi introducerea de la tastatură a noului conţinut.

Ştergerea datelor din tabele

Ştergerea datelor dintr-o tabelă se poate realiza la două niveluri: unul logic şi unul fizic. Ştergerea logică a unei înregistrări constă, de fapt, în marcarea înregistrării astfel încât aceasta să fie invizibilă pentru comenzile de prelucrare. O înregistrare marcată pentru ştergere nu va fi luată în considerare de comenzile de prelucrare, chiar dacă ea există în tabela respectivă.

Obs *Pentru ca înregistrările marcate pentru ştergere să nu fie luate în considerare la prelucrare este necesar ca* SET DELETED *să fie* ON, *în cazul* OFF *ele fiind prelucrate ca şi înregistrările nemarcate.*

Marcarea pentru ştergere a unei înregistrări a tabelei din fereastra **Browse** se realizează printr-un clic simplu pe marcatorul de ştergere din dreptul înregistrării respective. Acest marcator este un comutator plasat într-o coloană specială din partea stângă a ferestrei.

Coloana marcatorilor de ştergere logică

Angajati

Marca	Nume	Prenume	Functia
101	Popa	Alexandru	Director gener
102	Toma	Dumitru	Director execu
103	Popescu	Ioana Maria	Contabil sef
104	Constantinescu	Viorica	Consilier 2
105	Vasiliu	Rodica	Secretara
106	Dimoftache	Constantin	Referent 3
107	Musat	Florica Violeta	Secretara
108	Petrina	Luminita	Inginer sef
109	Elefterescu	Carmen	Referent 2

Calea directă de marcare pentru ştergere este Ctrl+T, la a cărei acţionare se poziţionează marcatorul în dreptul înregistrării curente (în care este plasat cursorul).

Ştergerea fizică a înregistrărilor marcate logic înseamnă, de fapt, eliminarea acestora din fişierul tabelei. Această operaţie se declanşează prin alegerea opţiunii **Remove Deleted Records** (elimină înregistrările şterse) a meniului **Table**.

Diferenţa esenţială între ştergerea logică şi cea fizică este aceea că o înregistrare marcată pentru ştergere se poate reface, în sensul că se poate demarca, pe când o înregistrare ştearsă fizic nu mai poate fi refăcută prin nici o metodă.

Demarcarea unei înregistrări şterse logic (operaţie denumită şi „rechemare") se realizează simplu, prin deselectarea marcatorului din dreptul înregistrării din fereastra **Browse**.

Stabilirea automată a înregistrării curente. Deplasarea cursorului

Pentru a realiza o anumită operaţie asupra unei înregistrări a tabelei deschise în fereastra **Browse**, înregistrarea respectivă trebuie mai întâi selectată. Operaţia se poate realiza manual, prin deplasarea cursorului cu ajutorul tastelor direcţionale ↑ şi ↓. Dacă însă avem de-a face cu multe înregistrări, deplasarea secvenţială poate fi consumatoare de timp, motiv pentru care se foloseşte metoda automată (închipuiţi-vă ce înseamnă deplasarea cursorului peste 100000 de înregistrări). Cu ajutorul metodei automate putem sări direct la înregistrarea dorită, fără a mai fi nevoie de parcurgerea înregistrărilor intermediare.

Deplasarea rapidă este realizată cu ajutorul opţiunilor submeniului afişat la selectarea opţiunii **Go to Record** (mergi la înregistrarea) a meniului **Table**. Acestea au următoarele semnificaţii:

- **Top** (în vârf) – pentru selectarea primei înregistrări a tabelei;
- **Bottom** (la bază) – pentru selectarea ultimei înregistrări a tabelei;
- **Next** (următoarea) – pentru deplasarea la următoarea înregistrare a tabelei (după cea curentă). Este echivalentă cu acţionarea tastei ↓;
- **Previous** (anterioara) – pentru deplasarea la înregistrarea anterioară a tabelei (înainte de cea curentă). Este echivalentă cu acţionarea tastei ↑;
- **Record #...** (înregistrarea nr...) – pentru deplasarea la înregistrarea cu numărul...;
- **Locate...** (căutare) – pentru deplasarea la prima înregistrare care respectă o anumită condiţie.

Primele patru opţiuni prezentate mai sus realizează o deplasare dependentă de starea de ordonare a tabelei respective. Aceasta înseamnă că, dacă o tabelă este

ordonată (prin indexare), prima înregistrare a tabelei va fi alta decât în cazul în care tabela ar fi neordonată. Opţiunile respective ţin cont de ordinea logică a înregistrărilor.

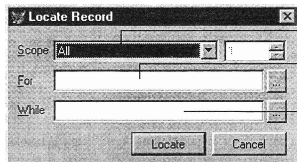
Exemplu

*De exemplu, alegerea opţiunii **Bottom** determină plasarea cursorului în ultima dintre înregistrările tabelei, în ordinea logică a acestora (dată de indexul activ).*

În schimb, opţiunea **Record #...** realizează o deplasare dată de ordinea fizică a înregistrărilor tabelei, indiferent dacă la momentul respectiv tabela era sau nu ordonată logic prin indexare. Cu alte cuvinte, alegând această opţiune, se va plasa cursorul în înregistrarea cu numărul specificat, indiferent în ce poziţie apare ea ca urmare a ordonării logice prin indexare.

Stabilirea automată a înregistrării curente. Căutări în tabelă

Ultima opţiune prezentată anterior, **Locate...** (căutare), este folosită pentru căutarea unor date în tabelă. O dată găsite datele respective, cursorul se poziţionează pe înregistrarea care le conţine. Prin urmare, acest tip de deplasare a cursorului este stabilit de sistem.



Aici se stabileşte domeniul înregistrărilor de parcurs

Aici se introduce criteriul de căutare de tip „caută până când..."

Aici se introduce criteriul de căutare de tip „caută atâta timp cât..."

Căutarea unor date într-o tabelă se face în funcţie de un anumit criteriu. La căutare pot fi parcurse toate înregistrările tabelei sau numai o parte a acestora, lucru stabilit prin domeniul înregistrărilor de parcurs (a se vedea următorul paragraf).

În cazul în care căutarea se termină cu succes, indicatorul de înregistrări va fi poziţionat pe înregistrarea găsită. Dacă însă nu se găseşte nici o înregistrare care să respecte condiţiile impuse, indicatorul de înregistrări al tabelei va arăta ultima înregistrare.

Exemplu

Să presupunem că dorim căutarea în tabela ANGAJATI *a unui director cu numele* Vasilescu. *Pentru aceasta, se alege din lista* **Scope** *a ferestrei* **Locate Record** *domeniul* All *(se vor parcurge toate înregistrările). În câmpul de editare* **For** *se introduce condiția ce trebuie respectată, adică:*

```
SUBSTR(Angajati.functie,1,3)=="Dir" AND;
  SUBSTR(Angajati.nume,1,LEN("Vasilescu"))=="Vasilescu"
```

Prelucrarea grupurilor de înregistrări

Unele comenzi de prelucrare a datelor din tabele pot acționa asupra mai multor înregistrări. Pentru aceasta este necesar un mecanism de selecție a înregistrărilor respective, mecanism ce va fi discutat în continuare.

O primă selecție privind înregistrările care urmează a fi prelucrate de către o comandă se realizează prin intermediul „domeniului înregistrărilor". Acesta poate fi:

- *All* (toate) – în acest caz, în prelucrare vor intra toate înregistrările tabelei;
- *Next...* (următoarele...) – pentru prelucrare vor fi alese următoarele *n* înregistrări de după cea curentă (inclusiv), în ordinea logică a acestora (dată de indexul activ);
- *Record...* (înregistrarea...) – ca urmare a selectării acestei opțiuni, va fi prelucrată înregistrarea cu numărul specificat (este luată în considerare ordinea fizică a înregistrărilor în tabelă);
- *Rest* (restul) – vor fi selectate pentru prelucrare toate înregistrările începând de la cea curentă, inclusiv, până la sfârșitul tabelei.

Obs

Variantele Next *și* Rest *depind de poziția curentă a indicatorului de înregistrări, pe când celelalte două alternative,* All *și* Record, *nu.*

Înregistrările tabelei sunt prelucrate în ordinea lor logică (dată de indexul activ), excepție făcând cazul Record, *când este prelucrată numai înregistrarea cu numărul de ordine (fizic) respectiv.*

O condiție logică de selecție a înregistrărilor ce urmează a fi prelucrate este precizată cu ajutorul clauzei **For** (pentru). Această clauză este însoțită de o expresie logică, ce se evaluează pentru fiecare înregistrare din domeniul specificat. Dacă valoarea obținută în urma evaluării este *adevărat*, înregistrarea va fi prelucrată, iar în caz contrar, ea va fi omisă de la prelucrare.

Obs

*Condiția **For** impusă înregistrărilor pentru a fi prelucrate se suprapune clauzei domeniului (*All, Next, Record *sau* Rest*). Cu alte cuvinte, condiția **For** va fi evaluată numai pentru înregistrările din domeniul specificat. Pot exista înregistrări care să respecte condiția **For**, dar să nu se afle în domeniul specificat – acestea nu vor face obiectul prelucrării.*

Exemplu

Să presupunem că dorim ca din tabela `ANGAJATI` *să selectăm pentru prelucrare următoarele 24 de înregistrări, iar dintre acestea numai pe cele care au în câmpul* `functie` *codul* `"Sec"` *(numai secretarele). Pentru aceasta, în comanda de prelucrare vom folosi:*

Next *24*

For `Angajati.functie=="Sec"`

Un alt tip de condiție impusă înregistrărilor pentru a fi prelucrate este **While** (atâta timp cât). Ca și **For**, această clauză este însoțită de o condiție care este evaluată pentru toate înregistrările domeniului specificat. Dar spre deosebire de criteriul de selecție **For**, conform criteriului **While** vor fi selectate înregistrări atâta timp cât condiția specificată este evaluată la valoarea *adevărat*. Prima înregistrare întâlnită care nu respectă condiția respectivă va determina sfârșitul căutării. Înregistrările următoare nu vor fi prelucrate, chiar dacă ele respectă condiția impusă.

Exemplu

Având tabela `ANGAJATI`*, ordonată alfabetic după numele angajaților, să presupunem că dorim prelucrarea tuturor înregistrărilor corespunzătoare salariaților al căror nume începe cu A (indicatorul de înregistrări se află la începutul tabelei). Condiția impusă este:*

All

While `SUBSTR(Angajati.nume,1,1)=='A'`

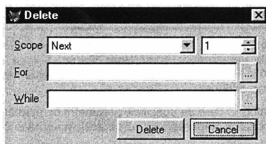
O dată întâlnit primul salariat care nu respectă condiția impusă (numele său nu începe cu A), prelucrarea se va termina.

Obs

*Dacă pentru o prelucrare se impune și o condiție de tip **For** și una de tip **While**, prioritate are condiția **While**.*

Să vedem acum câteva tipuri de operații care pot acționa asupra unor grupuri de înregistrări. Una este ștergerea logică sau marcarea pentru ștergere. Pentru a activa marcatorul de ștergere al mai multor înregistrări se alege opțiunea **Delete Records**

(ştergere înregistrări) a meniului **Table**, operaţie ce are ca efect afişarea unei ferestre de dialog în care pot fi specificate condiţiile pentru selecţia înregistrărilor de şters.



Deselectarea înregistrărilor marcate pentru ştergere reprezintă de asemenea o operaţie ce poate acţiona asupra mai multor înregistrări odată. Opţiunea corespunzătoare a meniului **Table** este **Recall Records** (rechemare înregistrări).

Exemplu

Să presupunem că în tabela angajaţilor dorim să marcăm pentru ştergere toţi angajaţii cu vechime mai mare de 2 ani, dar nu şi pe cei care au fost angajaţi în lunile de vară.

Pentru aceasta, putem să ştergem mai întâi toţi angajaţii cu vechime mai mare de 2 ani, printr-o comandă de ştergere cu următoarele condiţii:

All

For `(DATE()-Angajati.data_ang)>(365*2)`

Urmează rechemarea salariaţilor şterşi anterior, care au fost angajaţi în lunile de vară, Iunie, Iulie sau August. Comanda de rechemare va trebui însoţită de următoarele condiţii:

All

For `INLIST(MONTH(Angajati.data_ang),6,7,8)`

Completarea automată a câmpurilor tabelei

Una dintre operaţiile de prelucrare în bloc a înregistrărilor unei tabele este completarea automată. Aceasta înseamnă că un anumit câmp poate fi completat cu o valoare obţinută prin evaluarea unei expresii specificate de utilizator.

Exemplu

Să presupunem că dorim creşterea cu 10 procente a salariului brut al tuturor angajaţilor din tabela **ANGAJATI** *cu o vechime mai mare de doi ani.*

Operaţia presupune specificarea câmpului tabelei care va fi completat automat şi a expresiei care va da valoarea de completare. Pentru fiecare înregistrare selectată (prin domeniu şi criteriile **For** şi **While**) va fi evaluată expresia respectivă şi câmpul se va completa cu valoarea obţinută.

Fereastra de dialog în care se specifică parametrii completării este **Replace Field** (înlocuire câmp), deschisă la alegerea opţiunii cu acelaşi nume a meniului **Table**.

Exemplu

Rezolvarea problemei propuse în exemplul anterior este dată mai jos.

Din lista **Field** *se selectează câmpul* **sal_brut**, *care reprezintă câmpul tabelei* **ANGAJATI** *care va fi completat automat.*

Expresia care dă valoarea de înlocuire este:

```
Angajati.sal_brut*1.10
```

Din lista **Scope** *se alege domeniul* All.

*Condiția **For** impusă înregistrărilor de prelucrat este:*

```
(DATE()-Angajati.data_ang)>(365*2)
```

Folosirea limbajului Visual FoxPro pentru prelucrarea datelor din bazele de date

Deschiderea și închiderea tabelelor și a bazelor de date

Pentru a avea acces la datele dintr-o bază de date, aceasta trebuie mai întâi deschisă. La fel se întâmplă și cu tabelele, simple sau încorporate în bazele de date. După terminarea lucrului cu tabelele și bazele de date, acestea se închid. Prin urmare, pentru prelucrarea datelor dintr-o bază de date vom folosi următoarea procedură:

```
<deschidere bază de date>
<deschidere tabele>
...
<operații de prelucrare>
...
<închidere tabele>
<închidere bază de date>
```

Se observă că mai întâi este deschisă baza de date, apoi tabelele conținute în ea. După operațiile de prelucrare respective se închid tabelele și baza de date. La deschiderea unei tabele legate, baza de date respectivă este deschisă automat și, de aceea, comanda de deschidere a bazei de date poate fi eliminată din cod. Comanda de închidere a bazei de date (`CLOSE DATABASES`) determină și închiderea tabelelor componente ale acesteia. Prin urmare, și comenzile de închidere de tabele pot fi eliminate. Cu toate acestea, pentru o mai bună lizibilitate, se preferă secvența de mai sus.

Deschiderea și închiderea unei baze de date

Înainte de folosirea unui element memorat într-o bază de date, aceasta trebuie deschisă. Deschiderea unei baze de date se face cu ajutorul comenzii `OPEN DATABASE`:

```
OPEN DATABASE <nume bază de date>
```

Extensia implicită a fişierului bazei de date este .DBC. Deschiderea unei baze de date nu implică şi deschiderea automată a tabelelor conţinute, operaţie care trebuie realizată explicit de proiectant.

Închiderea bazei de date curente se realizează cu comanda CLOSE DATABASES. Dacă în comandă se include şi clauza ALL, vor fi închise toate bazele de date deschise, toate tabelele legate sau izolate, precum şi alte tipuri de fişiere deschise în momentul respectiv.

Deschiderea şi închiderea unei tabele. Zone de lucru. Aliasul unei tabele

Deschiderea unei tabele (create anterior) se realizează cu ajutorul comenzii USE:

```
USE <tabelă>
```

După deschiderea unei tabele, acesteia i se atribuie un nume, sau „alias", prin intermediul căruia se face referire la ea şi la elementele sale.

Exemplu

De exemplu, deschiderea tabelei ANGAJATI şi afişarea conţinutului câmpului sal_brut al acestei tabele se poate face prin următoarea secvenţă de comenzi:

```
USE angajati
? angajati.sal_brut
```

Se observă modul în care se face referirea la câmpul tabelei, prin precedarea sa de către aliasul tabelei, angajati.

Pentru manipularea tabelelor, Visual FoxPro foloseşte zone speciale de memorie, numite „zone de lucru". Pentru identificarea zonelor de lucru se folosesc două metode:

- primele 10 zone de lucru se denumesc cu litere de la A la J, reprezentând primele 10 litere din alfabet;
- pentru toate zonele de lucru putem folosi pentru identificare numerele de la 1 la numărul maxim de zone de lucru (peste 30000).

Pentru a deschide o tabelă într-o anumită zonă de lucru se foloseşte clauza IN a comenzii USE:

```
USE <tabelă> IN <zonă de lucru>
```

Obs

La un moment dat, într-o zonă de lucru nu poate fi deschisă decât o tabelă. Deschiderea unei tabele într-o zonă de lucru ocupată trebuie precedată de închiderea tabelei din zona respectivă.

Dacă în locul zonei de lucru a clauzei IN se foloseşte valoarea 0, atunci tabela respectivă va fi deschisă în prima zonă de lucru neocupată. Această variantă este des folosită, deoarece lasă în seama sistemului sarcina gestiunii zonelor de lucru. Nu interesează în ce zonă de lucru a fost deschisă o tabelă, deoarece pentru a ne referi la elementele sale vom folosi aliasul.

Exemplu

De exemplu, dacă trebuie să folosim două tabele simultan, deschiderea lor se poate face printr-o secvenţă de tipul:

```
USE <tabelă 1> IN 0
USE <tabelă 2> IN 0
```

Deschiderea unei tabele deja deschise (într-o altă zonă de lucru) este posibilă numai dacă în comanda USE de deschidere se foloseşte clauza AGAIN. Deschiderea de mai multe ori a unei tabele se foloseşte în cazuri rare, când se doreşte efectuarea unor prelucrări speciale.

De obicei, aliasul unei tabele este stabilit automat de către sistem (de cele mai multe ori, este acelaşi cu numele fişierului tabelei, fără extensie). Dacă se doreşte atribuirea unui anumit alias, trebuie utilizată în comanda USE clauza ALIAS, urmată de aliasul dorit.

Exemplu

De exemplu, putem deschide tabela ANGAJATI, *la care să facem referire prin identificatorul* salariat.

```
USE angajati IN 0 ALIAS salariat
? Salariat.sal_brut
```

Din totalul zonelor de lucru disponibile, una singură este la un moment dat curentă (sau activă). O comandă de prelucrare pentru care nu se specifică în mod explicit tabela la care se referă va opera asupra tabelei active, cea deschisă în zona de lucru curentă. Schimbarea zonei de lucru active se realizează cu comanda SELECT, care trebuie urmată de numele zonei de lucru sau de aliasul tabelei deschise în aceasta.

Exemplu

Deschiderea unei tabele în zona de lucru B se poate face în două moduri:

```
USE <tabelă> IN B
```

sau

```
SELECT B
USE <tabelă>
```

Închiderea unei tabele se realizează tot cu ajutorul comenzii USE, în care însă nu se specifică nici un nume de tabelă, ci, eventual, doar numele zonei de lucru în care aceasta este deschisă (după clauza IN). Închiderea tabelelor se poate realiza şi cu ajutorul comenzilor CLOSE ALL şi CLOSE DATABASES.

Indicatorul de înregistrări. Înregistrarea curentă

Prelucrarea datelor dintr-o tabelă se face la nivel de înregistrare, în sensul că la un moment dat este prelucrată o singură înregistrare. Pentru evidenţa înregistrării care urmează a fi prelucrată, sistemul foloseşte o variabilă specială numită *„indicatorul de înregistrări"*. Aceasta este specifică tabelei deschise şi deci vom avea atâtea indicatoare de înregistrări câte zone de lucru sunt ocupate cu tabele.

Există o mulţime de comenzi şi funcţii care prelucrează anumite înregistrări din tabelă, identificarea acestora făcându-se prin intermediul înregistrării curente.

Exemplu

De exemplu, comanda:

```
DISPLAY NEXT 2
```

afişează următoarele două înregistrări din tabela activă, începând de la înregistrarea curentă, inclusiv.

Aflarea înregistrării curente dintr-o tabelă, deci a conţinutului indicatorului de înregistrări, se face folosind funcţia RECNO():

```
RECNO(<tabelă>)
```

Aceasta returnează o valoare numerică reprezentând numărul înregistrării curente din tabela specificată.

Exemplu

```
USE angajati
? RECNO("angajati")
```

Comenzile folosite pentru schimbarea înregistrării curente (şi deci pentru deplasarea într-o tabelă) sunt GOTO şi SKIP. Prima dintre ele poziţionează indicatorul de înregistrări pe o anumită înregistrare dintr-o tabelă.

```
GOTO <număr înreg.> IN <tabelă>
GOTO TOP IN <tabelă>
GOTO BOTTOM IN <tabelă>
```

Prima formă a comenzii se foloseşte pentru plasarea indicatorului de înregistrări pe o înregistrare din interiorul tabelei, cea cu numărul specificat. Formele următoare ale comenzii se folosesc pentru poziţionarea indicatorului de înregistrări la extremele tabelei, şi anume: clauza TOP pentru poziţionarea pe prima înregistrare a tabelei, iar clauza BOTTOM pentru plasarea pe ultima înregistrare.

În cazul ultimelor două forme ale comenzii GOTO este luată în considerare ordinea logică a înregistrărilor tabelei, adică cea dată de indexul activ.

Exemplu

```
USE angajati
GOTO 2            && inregistrarea curenta va fi 2
? RECNO ()
2
GOTO RECNO()+1    && pozitionarea pe inregistrarea urmatoare;
                     (inregistrarea curenta + 1)
GOTO TOP          && pozitionare pe prima inregistrare
? RECNO ()
1
GOTO BOTTOM       && pozitionare pe ultima inregistrare
? RECNO ()
452
USE
```

Un alt tip de deplasare de-a lungul tabelei (cu indicatorul de înregistrări) este realizat cu ajutorul comenzii SKIP. Aceasta mută indicatorul peste un număr precizat de înregistrări, relativ la înregistrarea curentă. Deplasarea ţine cont de ordinea logică a înregistrărilor tabelei.

```
SKIP <număr de înreg.> IN <tabelă>
```

Numărul de înregistrări peste care se sare poate fi atât pozitiv (se sare spre o înregistrare cu un număr mai mare), cât şi negativ (caz în care se sare spre o înregistrare anterioară celei curente).

Dacă se sare peste ultima înregistrare a tabelei, indicatorul de înregistrări va conţine numărul de înregistrări din tabelă plus 1, iar funcţia EOF() (care indică dacă s-a atins sfârşitul fişierului) va returna valoarea *adevărat*. În cazul saltului înaintea primei înregistrări, funcţia BOF() (care indică dacă s-a atins începutul fişierului) va returna valoarea *adevărat*.

Exemplu

```
USE angajati
? RECNO ()
1
SKIP 2
```

```
? RECNO ()
3
SKIP -1
? RECNO ()
2
USE
```

Numărul de înregistrări dintr-o tabelă este returnat de funcția `RECCOUNT()`, căreia i se transmite ca parametru aliasul tabelei la care se referă.

Prelucrarea grupurilor de înregistrări. Domeniul înregistrărilor

Limbajul FoxPro include comenzi cu ajutorul cărora sunt prelucrate mai multe înregistrări ale unei tabele, alegerea acestora făcându-se printr-o serie de condiții de selecție. Mulțimea înregistrărilor selectate formează *„domeniul înregistrărilor"* la care se referă comanda respectivă.

Pentru precizarea domeniului înregistrărilor asupra cărora acționează o comandă se folosesc clauzele `<domeniu>`, `FOR` și `WHILE`. Prima dintre clauze, `<domeniu>`, se va înlocui cu una din următoarele construcții:

- `ALL` – pentru selectarea tuturor înregistrărilor din tabelă;
- `NEXT n` – atunci când comanda se referă la următoarele *n* înregistrări, începând de la înregistrarea curentă, inclusiv;
- `RECORD n` – când comanda trebuie să opereze numai asupra înregistrării cu numărul *n*;
- `REST` – pentru selectarea înregistrărilor începând de la cea curentă inclusiv și până la sfârșitul tabelei.

Clauza `FOR` (pentru) este urmată de o condiție care se evaluează pentru fiecare înregistrare în parte (din domeniul specificat), prelucrarea urmând a avea loc numai pentru acele înregistrări pentru care expresia logică respectivă are valoarea *adevărat*.

Clauza `WHILE` (atâta timp cât) este asemănătoare cu `FOR`, selectarea înregistrărilor făcându-se tot în funcție de expresia logică inclusă (pentru valoarea *adevărat* a acesteia). Însă, spre deosebire de clauza `FOR`, care, după găsirea unei înregistrări ce nu respectă condiția, continuă testarea celorlalte, clauza `WHILE` întrerupe testarea înregistrărilor când găsește o înregistrare ce nu respectă condiția dată.

Obs

Dacă se specifică ambele clauze, `FOR` şi `WHILE`, prima care contează este clauza `WHILE`.

Folosind clauza `FOR`, viteza de prelucrare creşte foarte mult şi, de aceea, se recomandă utilizarea acesteia oricând este posibil.

În expresia logică din clauzele `FOR` şi `WHILE` trebuie să intervină o mărime ce variază în funcţie de înregistrare. Aceasta trebuie să depindă fie de numărul înregistrării, fie de conţinutul acesteia.

Exemplu

Un domeniu de forma:

```
i=4
... ALL FOR i=2 ...
```

este egal cu 0 înregistrări, deoarece, pentru orice înregistrare, i este diferit de 2, deci expresia logică va fi falsă. În mod asemănător, domeniul:

```
ALL WHILE 1+1=2
```

reprezintă toate înregistrările din tabelă (1+1 este egal cu 2 pentru toate înregistrările).

Construcţia:

```
ALL FOR RECNO()>3
```

are ca efect selectarea tuturor înregistrărilor începând de la înregistrarea 3, exclusiv.

Observăm în acest ultim exemplu, spre deosebire de celelalte două, că în condiţia logică a clauzei `FOR` intervine funcţia `RECNO()` (reprezentând numărul înregistrării), care este dependentă de înregistrarea de testat.

Adăugarea de înregistrări la o tabelă

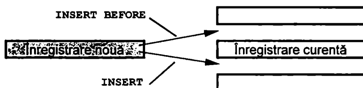
Adăugarea de înregistrări la o tabelă se poate face în două moduri, în funcţie de poziţia pe care o va ocupa noua înregistrare:

- adăugarea de înregistrări noi la sfârşitul tabelei;
- introducerea de înregistrări noi în interiorul tabelei.

Prima metodă este implementată prin intermediul comenzii APPEND BLANK, care are ca efect adăugarea unei înregistrări goale la sfârşitul tabelei (urmând ca informaţia utilă să se încarce mai târziu, prin alte comenzi).

Cea de-a doua metodă de adăugare a unei înregistrări noi la o tabelă o reprezintă inserarea înregistrării în interiorul tabelei, folosindu-se comanda INSERT BLANK. Comanda realizează introducerea unei înregistrări goale imediat după înregistrarea curentă (sau imediat înaintea acesteia, dacă se adaugă la comandă clauza BEFORE). Prin urmare, înainte de introducerea unei înregistrări în interiorul tabelei, trebuie poziţionat corespunzător indicatorul de înregistrări.

Tehnica de inserare este ilustrată în figura următoare:



Exemplu

```
USE angajati
GOTO 2                  && se pozitioneaza indicatorul de;
                           inregistrari pe inregistrarea 2
INSERT BLANK BEFORE     && se insereaza o noua inregistrare;
                           in pozitia 2
USE
```

Afişarea conţinutului unei tabele

În cadrul unui sistem informatic, metoda recomandată pentru afişarea conţinutului uneia sau mai multor tabele este în general cea a rapoartelor construite cu un utilitar special al sistemului, Constructorul de rapoarte. O metodă simplistă de realizare a unor situaţii pe baza unei tabele, preluată din versiunile FoxPro mai vechi, este cea a comenzilor LIST şi DISPLAY. Această metodă se poate folosi mai ales în timpul proiectării, când se doreşte afişarea conţinutului unei tabele, fără a interesa în mod deosebit aspectul rezultatelor obţinute.

Comanda LIST afişează conţinutul tabelei active.

```
LIST FIELDS <listă câmpuri>
  <domeniu> FOR <condiţie> WHILE <condiţie>
  OFF
  TO PRINTER | TO FILE <fişier>
  NOCONSOLE
```

În mod implicit, comanda determină afişarea tuturor câmpurilor tabelei. Clauza FIELDS se foloseşte pentru afişarea doar a anumitor câmpuri, cele specificate în lista care o însoţeşte (ordinea în listă dă şi ordinea afişării câmpurilor).

Exemplu

```
USE angajati
LIST FIELDS nume, prenume, functie
NOTE se afiseaza doar campurile din lista
USE
```

<domeniu>, FOR şi WHILE determină domeniul înregistrărilor ce vor fi afişate cu comanda LIST. Dacă aceste clauze lipsesc, se vor afişa toate înregistrările, acesta fiind domeniul implicit pentru comanda LIST.

Exemplu

```
USE angajati
LIST FOR sal_brut>1350000
NOTE se afiseaza doar salariatii care au salariile;
      peste 1350000;
USE
```

Locul afişării este stabilit prin intermediul clauzelor NOCONSOLE, TO PRINTER şi TO FILE. În mod implicit, afişarea se realizează pe ecran, adică în fereastra sistemului. Pentru inhibarea afişării pe ecran se foloseşte clauza NOCONSOLE. Această clauză este utilizată, în special, când se afişează la imprimantă sau într-un fişier pe disc şi nu se doreşte modificarea conţinutului ecranului.

În paralel cu afişarea pe ecran (sau în lipsa acesteia, când se foloseşte NOCONSOLE) se poate face şi afişarea la imprimantă, dacă se foloseşte clauza TO PRINTER, sau într-un fişier, când se utilizează clauza TO FILE.

Exemplu

Afişarea doar la imprimantă a conţinutului tabelei ANGAJATI *se face cu următoarea secvenţă de instrucţiuni:*

```
USE angajati
LIST NOCONSOLE TO PRINTER
USE
```

Pentru a obţine conţinutul tabelei ANGAJATI *atât în fişierul* LISTARE.TXT *cât şi pe ecran, se foloseşte secvenţa:*

```
USE angajati
LIST TO FILE listare.txt
USE
```

Comanda DISPLAY este asemănătoare cu LIST, existând însă şi o serie de diferenţe, dintre care cea mai importantă este domeniul implicit: la LIST toate înregistrările (ALL), pe când la DISPLAY doar înregistrarea curentă (NEXT 1).

Obs

Comanda LIST este echivalentă, din punct de vedere al domeniului înregistrărilor, cu DISPLAY ALL, iar comanda DISPLAY este echivalentă cu comanda LIST NEXT 1.

Modificarea conţinutului unei tabele

Modificarea prin cod a datelor stocate într-o tabelă se realizează cu comanda REPLACE:

```
REPLACE
  <câmp1> WITH <valoare1>,
  <câmp2> WITH <valoare2>,
  ...
  <domeniu> FOR <condiţie> WHILE <condiţie>
```

Ea înlocuieşte vechea valoare din câmpurile specificate (<câmp1>, <câmp2>,...) cu valorile corespunzătoare (<valoare1>, <valoare2>,...).

<domeniu>, FOR şi WHILE indică domeniul înregistrărilor la care se referă comanda REPLACE, domeniul implicit fiind înregistrarea curentă.

Exemplu

La tabela ANGAJATI se va adăuga o nouă înregistrare, cu următorul conţinut:

```
NUME: Toma
PRENUME: Constantin
SAL_BRUT: 1456000
...
```

Secvenţa de comenzi care realizează acest lucru este:

```
USE angajati
APPEND BLANK
REPLACE nume WITH 'Toma',;
        prenume WITH 'Constantin',;
        sal_brut WITH 1456000,;
        ...
LIST
USE
```

Tehnica de modificare a conţinutului unei tabele, prezentată anterior, preia datele sursă fie direct din program, fie din variabile de memorie. Fiecare câmp care se doreşte a fi modificat trebuie precizat în comanda REPLACE. O altă tehnică de modificare a conţinutului înregistrării curente a unei tabele este dată de comenzile SCATTER şi GATHER. Prin intermediul acestor două comenzi se poate realiza transferul între înregistrarea curentă a tabelei şi un tablou sau un set de variabile. Comenzile acţionează asupra întregii înregistrări, deci asupra tuturor câmpurilor tabelei, fără a fi necesară specificarea lor explicită.

Comanda GATHER realizează transferul de la tablou sau de la setul de variabile la tabelă, iar SCATTER realizează transferul invers. Comanda GATHER poate avea două forme:

```
GATHER FROM <tablou>
GATHER MEMVAR
```

Prima formă se foloseşte pentru a încărca datele din tabloul specificat în înregistrarea curentă, iar cea de-a două formă foloseşte ca sursă de date un set special de variabile, cu aceleaşi nume ca şi câmpurile tabelei. Accesul la setul de variabile (creat cu comanda SCATTER cu clauza MEMVAR) se face prin construcţia:

```
m.<nume variabilă>
```

Comanda GATHER poate conţine clauza FIELDS, urmată de lista câmpurilor care se vor copia în înregistrarea curentă a tabelei active. Clauza MEMO a comenzii GATHER se foloseşte atunci când tabela are un câmp de tip „memo" care se doreşte a fi şi el copiat (în lipsa clauzei, acesta este omis).

În cazul transferului dintr-un tablou, copierea se va face în felul următor: primul element al tabloului va fi copiat în primul câmp (eventual, din lista de câmpuri specificată) al tabelei active, al doilea element al tabloului în cel de-al doilea câmp şi aşa mai departe, până când se termină elementele tabloului sau câmpurile tabelei.

Opusă comenzii GATHER este comanda SCATTER, care copiază câmpurile înregistrării curente din tabela activă într-un tablou sau într-un set de variabile. Comanda are următoarele forme:

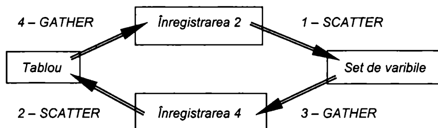
```
SCATTER TO <tablou>
SCATTER MEMVAR
```

Şi comanda SCATTER poate conţine clauzele FIELDS şi MEMO, cu aceeaşi utilizare. În plus, în comanda SCATTER poate fi inclusă clauza BLANK, ce are ca efect crearea tabloului specificat sau a setului de variabile asociat tabelei şi încărcarea acestora cu valori nule.

Exemplu

Vom considera tabela ANGAJATI, *în care dorim schimbarea între ele a înregistrărilor cu numerele 2 şi 4. La această interschimbare vom folosi ca intermediar un set de variabile cu aceleaşi nume cu ale câmpurilor corespunzătoare şi un tablou* v.

Tehnica de schimbare este indicată în următoarea figură:



Programul ce îndeplineşte această sarcină este următorul:

```
USE angajati
LIST FOR RECNO()=2 AND RECNO()=4
GOTO 2
SCATTER MEMVAR MEMO
NOTE se realizeaza operatia 1, copierea;
      inregistrarii 2 in setul de variabile
GOTO 4
SCATTER TO v MEMO
NOTE operatia 2: copierea inregistrarii 4 in tabloul v
GATHER FROM v MEMO
NOTE operatia 3: copierea setului de variabile in;
      inregistrarea 4
GOTO 2
GATHER MEMVAR MEMO
NOTE operatia 4: copierea tabloului in inregistrarea 2
LIST FOR RECNO()=2 AND RECNO()=4
USE
RELEASE ALL
```

Ştergerea înregistrărilor dintr-o tabelă

Alături de adăugare şi modificare, ştergerea reprezintă una din principalele operaţii în lucrul cu tabelele. Ştergerea unei înregistrări dintr-o tabelă se poate realiza la două niveluri, şi anume:

- *la nivel logic*, când înregistrarea nu este propriu-zis ştearsă din tabelă, ci este marcată într-un anumit mod („*marcată pentru ştergere*"), indicând astfel această stare a înregistrării. Există comenzi şi funcţii FoxPro (marea lor majoritate) care, înainte de accesul la o înregistrare, testează marcajul de ştergere al acesteia şi, în funcţie de ceea ce găsesc, consideră înregistrarea absentă sau prezentă în fişier;
- *la nivel fizic*, când înregistrarea este ştearsă efectiv din tabelă, ea neputând fi în nici un fel utilizată sau refăcută.

Marcarea pentru ştergere a uneia sau a mai multor înregistrări se face cu ajutorul comenzii `DELETE`:

```
DELETE
  <domeniu> FOR <condiţie> WHILE <condiţie>
```

`<domeniu>`, `FOR` şi `WHILE` identifică înregistrările ce vor fi marcate pentru ştergere. Domeniul implicit al comenzii `DELETE` este înregistrarea curentă.

Accesul la înregistrările marcate pentru ştergere este controlat de comanda `SET DELETED`. În starea `ON`, înregistrările marcate pentru ştergere nu vor fi accesibile celorlalte comenzi FoxPro. În starea `OFF` însă, înregistrările sunt accesibile indiferent de marcajul lor de ştergere.

Comenzile care acţionează asupra unei singure înregistrări sau care au ca domeniu implicit înregistrarea curentă nu sunt afectate de această comandă.

Exemplu

```
USE angajati
CLEAR
SET DELETED OFF
DELETE FOR MOD(RECNO(),2)=0
NOTE se sterg inregistrarile cu numar de ordine par
LIST
NOTE toate inregistrarile din tabela sunt afisate;
      cele sterse avand un asterisc in dreptul lor.
GOTO 2
DISPLAY && inregistrarea este afisata chiar daca;
      este marcata pentru stergere, DISPLAY avand ca;
      domeniu implicit inregistrarea curenta
USE
```

Exemplu

Pentru a vedea efectul ştergerii unor înregistrări asupra comenzii LIST*, vom folosi următorul exemplu:*

```
USE angajati
DELETE FOR MOD(RECNO(),2)=0
SET DELETED OFF
LIST FOR DELETED()=.F.
USE
```

acest exemplu fiind echivalent cu:

```
USE angajati
DELETE FOR MOD(RECNO(),2)=0
SET DELETED ON
LIST
USE
```

Observăm că la prima comandă LIST*, înregistrările marcate pentru ştergere sunt accesibile comenzii, pe când la cea de-a doua, înregistrările nu mai pot fi utilizate de comandă.*

În interiorul unui program, testarea marcajului de ştergere al unei înregistrări se face cu funcţia DELETED(), care returnează valoarea logică *adevărat* dacă înregistrarea curentă este marcată pentru ştergere şi *fals* în caz contrar.

Exemplu

```
USE angajati
SET DELETED OFF
DELETE RECORD 2
GOTO 2
? DELETED ()
.T.
GOTO 3
? DELETED ()
.F.
USE
```

În cazul când funcţia se referă la o altă tabelă decât cea activă, aceasta trebuie transmisă funcţiei ca parametru.

O înregistrare marcată pentru ştergere nu este ştearsă fizic din tabelă. Eliminându-se marcajul de ştergere, înregistrarea este refăcută, ea redevenind accesibilă tuturor comenzilor. Înlăturarea marcajului de ştergere (operaţie numită şi „*rechemare*") se realizează cu comanda RECALL, ce are o utilizare asemănătoare cu DELETE.

Exemplu

```
USE angajati
DELETE FOR RECNO()<=3;
    && se sterg primele 3 inregistrari
LIST          && se observa efectul stergerii
RECALL ALL    && sunt refacute toate inregistrarile.;
               Cele care nu erau marcate pentru;
                      stergere nu sunt afectate
LIST
USE
```

Până acum am prezentat ştergerea logică a înregistrărilor, adică marcarea lor pentru ştergere. Pentru ca o înregistrare să fie eliminată fizic din tabelă, ea trebuie ştearsă la nivel fizic. Pentru ştergerea la nivel fizic se foloseşte comanda PACK.

Obs *După aplicarea comenzii* PACK *asupra unei tabele, înregistrările nu mai pot fi refăcute; ştergerile sunt permanente.*

O ultimă comandă cu privire la ştergerea înregistrărilor din tabelă este comanda ZAP. Aceasta şterge fizic toate înregistrările din tabela activă, fiind echivalentă cu secvenţa de instrucţiuni:

```
DELETE ALL
PACK
```

Ori de câte ori se doreşte golirea totală a unei baze de date, este recomandată folosirea comenzii ZAP, care este mult mai rapidă decât secvenţa anterioară de instrucţiuni.

Obs *Înainte de execuţia comenzii, dacă opţiunea* SET SAFETY *este* ON, *se va mai afişa un mesaj de confirmare a ştergerii înregistrărilor.*

Accesul la înregistrările tabelei. Filtrarea

Pe lângă cele două metode de ştergere a înregistrărilor, logică şi fizică, mai există o metodă de control a accesului la înregistrările unei tabele, situată ca putere de control al accesului la înregistrări între cele două metode prezentate anterior. Spre deosebire de ştergerea fizică a unei înregistrări, această metodă nu elimină fizic înregistrarea, ci doar blochează accesul la ea. Deci, din acest punct de vedere, metoda este mai slabă decât ştergerea fizică.

Comparând metoda cu ştergerea logică a unei înregistrări, constatăm că ea este mai puternică decât ştergerea logică. Astfel, pe când o înregistrare marcată pentru

ştergere este accesibilă anumitor comenzi sau funcţii FoxPro (cum ar fi LIST), blocarea accesului la o înregistrare prin această metodă este valabilă pentru toate comenzile şi funcţiile FoxPro care folosesc tabela.

Metoda aceasta de control al accesului este implementată prin comanda SET FILTER:

```
SET FILTER TO <condiţie>
```

Ca efect al comenzii, în tabelă vor apărea doar înregistrările care îndeplinesc condiţia specificată. SET FILTER TO, fără condiţie, face ca toate înregistrările din tabelă să fie accesibile, adică înlătură filtrul aplicat anterior tabelei.

Exemplu

Din tabela ANGAJATI se vor elimina (nu fizic) toţi bărbaţii (sex=.T.):

```
USE angajati
LIST
SET FILTER TO sex=.T.
LIST
USE
```

O caracteristică importantă a acestei metode de control al accesului la înregistrările unei tabele este aceea că accesibilitatea unei înregistrări este dependentă de conţinutul său, după cum se poate observa şi din exemplul anterior. Prin urmare, s-ar putea inhiba accesul la o înregistrare a tabelei doar prin modificarea conţinutului înregistrării respective.

Căutarea datelor în tabele

În cazul tabelelor cu un număr mare de înregistrări, una dintre operaţiile foarte utile este căutarea, adică identificarea unei înregistrări care respectă o anumită condiţie. Comanda folosită pentru aceasta este LOCATE:

```
LOCATE FOR <condiţie>
  <domeniu> WHILE <condiţie>
```

Ea caută prima înregistrare care respectă condiţia din clauza FOR. Domeniul înregistrărilor care se testează este dat de clauzele <domeniu> şi WHILE, cel implicit fiind ALL.

În caz de reuşită, adică la găsirea unei înregistrări care respectă condiţia clauzei FOR, indicatorul de înregistrări se va poziţiona pe înregistrarea respectivă, funcţia FOUND() (ce va fi prezentată mai jos) va returna valoarea *adevărat*, iar funcţia EOF() va returna valoarea *fals* (nu s-a ajuns la sfârşitul fişierului). În caz contrar, indicatorul de

înregistrări va fi poziţionat după ultima înregistrare (numărul total de înregistrări + 1), FOUND() va returna *fals*, iar EOF() va returna valoarea logică *adevărat*.

Într-o tabelă pot exista mai multe înregistrări ce respectă o condiţie dată. Prima dintre acestea va fi găsită folosind comanda LOCATE, iar următoarele vor fi găsite prin intermediul comenzii CONTINUE. Aceasta găseşte următoarea înregistrare care respectă condiţia specificată în ultima comandă LOCATE aplicată tabelei active. Testarea reuşitei sau nereuşitei căutării se face ca şi la comanda LOCATE, cu ajutorul funcţiilor RECNO(), FOUND() şi EOF().

Funcţia FOUND() este folosită pentru testarea rezultatului unei căutări într-o tabelă, returnând valoarea *adevărat* în cazul unei căutări încheiate cu succes şi valoarea *fals* în cazul unei căutări nereuşite. Funcţia poate primi ca argument numele tabelei care se testează.

Exemplu

Să se găsească primele două persoane de sex masculin din tabela ANGAJATI.

```
USE angajati
LOCATE FOR sex=.T.
? FOUND()
.T.
? EOF()
.F.
? RECNO()
3
CONTINUE
? FOUND()
.T.
? EOF()
.F.
? RECNO()
4
USE
```

Calcule statistice cu datele din tabele

Simpla consultare a tabelelor nu este totdeauna suficientă pentru evidenţierea unor aspecte referitoare la datele stocate. De exemplu, având tabela ANGAJATI în care sunt memorate informaţiile referitoare la salariaţii unei unităţi economice, dorim să aflăm ponderea salariului persoanelor cu studii superioare în fondul total de salarii al unităţii. Rezolvarea acestei probleme presupune efectuarea unor calcule statistice cu datele din tabelă, calcularea fondului total de salarii şi a fondului de salarii pentru persoanele cu studii superioare şi raportarea celor două valori, pentru obţinerea procentului solicitat. În

acest paragraf sunt descrise comenzile şi funcţiile prin care se pot executa astfel de calcule statistice.

Calcularea numărului de înregistrări care respectă o anumită condiţie se realizează cu ajutorul comenzii COUNT. Prin includerea clauzelor referitoare la domeniul înregistrărilor, <domeniu>, FOR şi WHILE, se arată condiţiile ce trebuie îndeplinite de înregistrări. Rezultatul se depune într-o variabilă specificată în clauza TO a comenzii.

Exemplu

Numărarea persoanelor al căror salariu brut depăşeşte 2000000 lei se face prin următoarea secvenţă de comenzi:

```
USE angajati
COUNT FOR sal_brut>2000000 TO nr_sal
? nr_sal
9
USE
```

Un alt tip de calcul ce se poate efectua cu datele dintr-o tabelă este reprezentat de însumarea valorii unor câmpuri numerice. Comanda folosită este SUM şi ea poate fi urmată de o listă de expresii care se vor evalua pentru fiecare înregistrare în parte. Rezultatele obţinute prin evaluare se vor însuma şi apoi vor fi depuse în variabilele corespunzătoare specificate în lista clauzei TO: suma valorilor primei expresii, pentru toate înregistrările selectate, va fi depusă în prima variabilă, a doua sumă în cea de-a doua variabilă şi aşa mai departe.

```
SUM <expresie1>, <expresie2>,... TO <var1>, <var2>,...
```

În comandă pot fi incluse clauze referitoare la domeniul înregistrărilor, cu ajutorul cărora să se stabilească ce înregistrări vor fi luate în considerare de comandă.

Exemplu

Având tabela ANGAJATI, să se calculeze procentul salariului persoanelor cu studii superioare din fondul total de salarii al unităţii economice.

```
USE angajati
SUM sal_brut ALL FOR studii=='S' TO sal_stud
SUM sal_brut ALL TO sal_tot
? sal_stud/sal_tot
0.26
USE
```

Media aritmetică a valorilor dintr-unul sau mai multe câmpuri se realizează cu ajutorul comenzii AVERAGE. Aceasta are o sintaxă asemănătoare cu SUM. Deosebirea constă în împărţirea rezultatului obţinut prin însumarea valorilor din câmp la numărul de

înregistrări prelucrate (definiţia mediei aritmetice: suma valorilor, împărţită la numărul valorilor însumate).

Exemplu

Să se calculeze salariul mediu al angajaţilor cu studii medii din tabela **ANGAJATI**.

```
USE angajati
AVERAGE sal_brut FOR studii=='M' TO sal_mediu
? sal-mediu
1146253.22
USE
```

Acelaşi lucru se poate realiza şi folosind comenzile SUM *şi* COUNT, *cu ajutorul cărora se poate simula comanda* AVERAGE.

```
USE angajati
SUM sal_brut FOR studii=='M' TO tot_sal_med
COUNT FOR studii=='M' TO nr_sal_med
medie=tot_sal_med/nr_sal_med
? medie
1146253.22
USE
```

O comandă mai complexă folosită pentru calculele statistice este CALCULATE. Comanda este urmată de o listă de expresii care sunt calculate pe baza datelor din tabelă. În cadrul acestor expresii pot fi incluse o serie de funcţii statistice, cu următoarele semnificaţii:

- `AVG(<expresie>)` – calculează media aritmetică a valorilor expresiei respective (pentru fiecare înregistrare), expresie care poate conţine câmpuri numerice ale tabelei;
- `CNT()` – returnează numărul de înregistrări prelucrate;
- `MAX(<expresie>)` – dintre toate valorile expresiei (evaluată pentru fiecare înregistrare) este returnată valoarea maximă;
- `MIN(<expresie>)` – dintre toate valorile expresiei (evaluată pentru fiecare înregistrare) este returnată valoarea minimă;
- `NPV(<expresie1>,<expresie2>,<expresie3>)` – calculează valoarea prezentă netă a unei serii de plăţi diminuate la o rată a dobânzii constantă. Prima expresie reprezintă rata dobânzii, cea de-a doua plata din seria de plăţi considerate, iar cea de-a treia valoarea iniţială a investiţiei;
- `STD(<expresie>)` – calculează deviaţia standard a valorilor expresiei specificate pentru înregistrările selectate:

$$\sqrt{(\overline{<\exp>^2} - \overline{<\exp>}^2)}$$

- `SUM(<expresie>)` – calculează suma valorilor expresiei;
- `VAR(<expresie>)` – calculează abaterea pătratică medie (deviația standard la pătrat):

$$\overline{<\exp>^2} - \overline{<\exp>}^2$$

Evident, comanda `CALCULATE` poate conține clauze referitoare la domeniul înregistrărilor de prelucrat.

Exemplu

Să presupunem că avem o tabelă în care am stocat rezultatele unei experiențe, valori numerice. Vom lua spre exemplu următoarea serie de valori: 13, 47, 35, 9, 89, 123, 75, depozitate în câmpul NUMAR al tabelei NUMERE.

```
USE numere
CALCULATE avg(numar) TO media
? 'Media numerelor este:', media
CALCULATE cnt() TO nr_inreg
? 'Numarul de valori este: ', nr_inreg
CALCULATE max(numar), min(numar), sum(numar);
      TO maxim, minim, suma
? 'Valoarea maxima este:', maxim
? 'Valoarea minima este:', minim
? 'Suma numerelor este:', suma
CALCULATE npv(0.1, numar, 100) TO val_p
? 'Valoarea prezenta este:',val_p
CALCULATE std(numar), var(numar) TO dev_std, ab_patr
? 'Deviatia standard este:', dev_std
? 'Abaterea patratica medie este:', ab_patr
USE
```

Avem următoarele echivalențe:

	echivalent cu
`CALCULATE AVG(<exp>) TO <var>`	`AVERAGE <exp> TO <var>`
`CALCULATE CNT() TO <var>`	`COUNT TO <var>`
`CALCULATE SUM(<exp>) TO <var>`	`SUM <exp> TO <var>`
`CALCULATE STD(<exp>) TO <var>`	`CALCULATE SQRT(VAR(<exp>)); TO <var>`

Ordonarea datelor din tabele

Să presupunem că avem o carte de telefon din care dorim să aflăm numărul de telefon al unei anumite persoane. Dacă această carte nu ar fi ordonată alfabetic, căutarea ar fi imposibilă pentru om şi grea pentru calculator. În cazul în care cartea de telefon este ordonată alfabetic, putem găsi telefonul dorit în câteva zeci de secunde, calculatorului fiindu-i necesar un timp de ordinul zecimilor de secundă. Observăm că în acest exemplu apare problema ordonării informaţiilor după anumite criterii, deci a ordonării tabelelor ce conţin datele respective.

Există două metode de ordonare a unei tabele, şi anume:

- *ordonarea fizică* a tabelei – realizată prin schimbarea înregistrărilor între ele după un anumit algoritm, până când acestea sunt în ordinea dorită. În acest caz se obţine o nouă tabelă care conţine aceleaşi înregistrări ca şi cea de la care s-a pornit, dar în ordinea dorită;
- *indexarea* unei tabele – aceasta însemnând crearea unui nou fişier, care conţine informaţiile cu privire la ordinea înregistrărilor tabelei. Tabela este văzută prin intermediul acestui fişier în ordinea dorită, fără ca înregistrările să se afle efectiv în ordinea respectivă. Şi în acest caz se obţine un fişier nou, fişierul index, dar acesta nu conţine înregistrările tabelei, ci memorează numai ordinea lor.

Vom prezenta cele două metode în cele ce urmează.

Sortarea tabelelor

Ordonarea fizică a unei tabele se realizează cu ajutorul comenzii SORT. Tabela care se creează în urma sortării se precizează în clauza TO.

```
SORT TO <tabelă destinaţie>...
```

Criteriul de ordonare este format din unul sau mai multe câmpuri, specificate în clauza ON.

```
SORT... ON <câmp1> /A sau /D,
           <câmp2> /A sau /D,...
```

Primul câmp specificat va da primul criteriu de ordonare. În cazul în care se găsesc două înregistrări cu aceeaşi valoare în câmpul respectiv, se ia în considerare cel de-al doilea câmp din listă şi aşa mai departe.

/A se foloseşte pentru o ordonare crescătoare după câmpul respectiv, iar /D pentru o ordonare descrescătoare.

Comanda poate conţine clauze referitoare la domeniul înregistrărilor, cu ajutorul cărora să se realizeze o selecţie a înregistrărilor ce vor fi preluate în noua tabelă. De asemenea, se poate realiza şi o selecţie la nivel de câmp, cu ajutorul clauzei FIELDS. Aceasta poate fi însoţită de o listă de câmpuri care urmează a fi copiate în tabela destinaţie.

Exemplu

Se ordonează tabela ANGAJATI, *cheia de ordonare fiind numele salariaţilor, iar ordinea crescătoare. Noua tabelă va purta numele* ANGAJ_S.

```
USE angajati
LIST
SORT TO angaj_s ON nume /A, prenume /A
USE angaj_s
LIST
USE
```

Indexarea tabelelor

Cea de-a doua metodă de ordonare a unei tabele o reprezintă indexarea. Aceasta presupune crearea unui fişier nou, numit fişier index asociat tabelei, în care se memorează ordinea înregistrărilor. Accesul la o anumită înregistrare se face prin intermediul fişierului index.

Să luăm următorul exemplu: o tabelă în care avem încărcate şapte tipuri de materialeşi unelte de construcţie, cu seria, cantitatea şi valoarea acestora:

MATERIAL.CDX	
Poziţie	Valoare
3	3000
1	9000
6	16000
2	42000
5	78000
7	110000
4	154000

MATERIAL.DBF			
Denumire	Serie	Cantitate	Valoare
ciment	B3	10	9000
tigla	C2	43	42000
lopeti	S5	100	3000
nisip	B11	73	154000
caramida	C4	88	78000
tabla	S44	3	16000
pietris	S10	24	110000

Indexarea acestei tabele după valoare, în ordine crescătoare, presupune crearea fişierului MATERIAL.CDX, în care se vor memora poziţiile înregistrărilor din tabelă, în ordinea dorită. Accesul la înregistrări se face prin intermediul fişierului index asociat.

Astfel, dacă dorim afişarea înregistrării a treia din tabela MATERIAL.DBF, sistemul citeşte valoarea memorată în poziţia a treia a fişierului index, adică 6, şi o tratează ca poziţie a înregistrării respective în tabelă. Deci se va afişa înregistrarea de pe poziţia 6 din tabela MATERIAL.DBF (după cum indică săgeţile din figură). Observăm că ordinea fizică a înregistrărilor din tabelă nu s-a modificat.

Modul de lucru cu o tabelă indexată este următorul:

- mai întâi trebuie creat fişierul index asociat tabelei, ocazie cu care se specifică şi criteriile de ordonare dorite pentru tabelă. Această etapă poartă numele de *„indexarea tabelei"*;
- când se doreşte folosirea tabelei indexate anterior, se deschide (folosind comanda USE, de exemplu) şi o dată cu ea se deschid şi fişierele index asociate, fie automat de către Visual FoxPro, fie manual de către utilizator;
- se realizează operaţiile dorite asupra tabelei (adăugare, modificare, ştergere, afişare etc.), înregistrările fiind văzute în ordinea dată de fişierul index activ sau de eticheta index activă. Modificarea conţinutului tabelei determină actualizarea automată a fişierelor index deschise pentru tabela respectivă, ordinea înregistrărilor fiind actualizată, de asemenea, la modificarea tabelei;
- după ce se termină lucrul cu tabela, aceasta se închide şi, o dată cu ea, se închid şi fişierele index asociate ei la deschidere.

Crearea fişierelor index se face de obicei la crearea tabelei, după cum s-a arătat în capitolul referitor la această operaţie. De asemenea, în capitolul respectiv sunt prezentate şi tipurile de fişiere index ce pot fi construite pentru o tabelă. În cele ce urmează ne vom ocupa cu precădere de utilizarea indecşilor în lucrul cu tabele.

O tabelă poate avea mai mulţi indecşi, dar numai unul va fi activ la un moment dat, acesta fiind numit **index activ**. Ordinea în care este parcursă tabela este dată de indexul activ.

Stabilirea indexului activ, adică a ordinii curente în care este văzută tabela, se realizează cu ajutorul comenzii SET ORDER:

```
SET ORDER TO <nume index> IN <tabelă>
```

Indexul care va da ordinea înregistrărilor din tabelă (fişier index simplu sau etichetă index într-un fişier index compus) va fi specificat în clauza TO a comenzii. De asemenea, comanda mai poate conţine clauza IN, care precizează tabela la care se face referire.

Pentru aflarea numelui indexului activ se folosește funcția ORDER():

```
ORDER(<tabelă>)
```

Rezultatul acestei funcții este de tip șir de caractere.

Pe lângă prezentarea datelor unei tabele într-o anumită ordine, indexarea are și alt avantaj major, și anume acela al regăsirii mult mai rapide a datelor. Căutarea unei anumite înregistrări într-o tabelă indexată se face cu comanda SEEK sau cu funcția cu același nume. Comanda este urmată de o valoare care este căutată printre valorile cheii de indexare ale fiecărei înregistrări.

```
SEEK <expr>
```

Dacă este găsită o asemenea înregistrare, indicatorul de înregistrări se va plasa pe aceasta, funcția FOUND() va returna valoarea *adevărat*, iar funcția EOF() va returna valoarea *fals*. În caz contrar, indicatorul de înregistrări se va poziționa după ultima înregistrare, FOUND() va returna *fals*, iar EOF() va returna valoarea *adevărat*.

Funcția este influențată de comanda SET NEAR. Dacă SET NEAR este în starea ON, în caz de căutare eșuată indicatorul de înregistrări se va plasa imediat după cea mai apropiată înregistrare (în sensul potrivirii valorii cheii de indexare cu valoarea expresiei din comanda SEEK). Indiferent de reușita căutării, funcția RECNO(0) va returna numărul de ordine al înregistrării celei mai asemănătoare.

Obs *Comanda SEEK este asemănătoare cu comanda LOCATE de la tabelele neindexate, dar este mult mai rapidă, datorită unor tehnici speciale de căutare în tabelele indexate.*

Exemplu

```
USE material
SET ORDER TO valoare
SEEK 78000
? FOUND ()
.T.
? EOF ()
.F.
? RECNO ()
5
? RECNO (0)
0
DISPLAY
USE
```

Funcţia SEEK() este asemănătoare comenzii cu acelaşi nume. Ea primeşte ca argument valoarea de căutat (care se compară cu valorile cheii de indexare active)

```
SEEK(<expresie>)
```

şi returnează o valoare logică în funcţie de rezultatul căutării: *adevărat* pentru cazul găsirii înregistrării şi *fals* pentru inexistenţa unei înregistrări cu cheia specificată.

Obs *Funcţia SEEK() înlocuieşte combinaţia dintre comanda SEEK şi funcţia FOUND().*

Exemplu

```
USE material
SET ORDER TO valoare
? SEEK(78000)
.T.
? RECNO ()
5
USE
```

Tipuri speciale de câmpuri. Câmpurile „memo" şi câmpurile „generale"

Câmpurile „memo"

Tipul „memo" al unui câmp este folosit pentru memorarea textelor ale căror lungimi diferă foarte mult de la o înregistrare la alta. De exemplu, dacă în tabela ANGAJATI, în care se memorează date referitoare la salariaţii unei unităţi economice, dorim memorarea domiciliilor persoanelor respective, pentru câmpul **adresa** este recomandat tipul „memo", ştiut fiind faptul că adresele diferă foarte mult ca lungime de la un caz la altul.

O tabelă care conţine cel puţin un câmp „memo" are asociat un fişier suplimentar în care sunt depuse datele conţinute în câmpurile „memo". Pentru a se putea identifica ce date din fişierul „memo" asociat aparţin unei anumite înregistrări din tabelă, pe poziţia câmpului considerat, în înregistrarea tabelei, se memorează un indicator spre datele respective (poziţia primului caracter al câmpului respectiv).

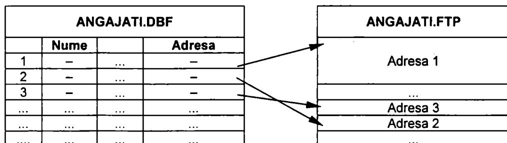
Accesul la conţinutul unui câmp memo se face în modul următor:

- se selectează înregistrarea dorită şi se citesc datele din câmpul „memo" respectiv, din tabelă;
- cu ajutorul acestor date se găseşte locul de depozitare a conţinutului câmpului „memo" în fişierul „memo" asociat;
- se citeşte din fişierul „memo", de la poziţia determinată anterior, conţinutul câmpului.

Încărcarea unor date într-un câmp „memo", la o anumită înregistrare, se face astfel:

- se găseşte un spaţiu liber în fişierul „memo" asociat tabelei, suficient pentru memorarea tuturor datelor, şi se încarcă aceste date în spaţiul respectiv;
- se completează în tabelă, la înregistrarea dorită, în câmpul „memo" respectiv, indicatorul necesar localizării în fişierul „memo" a conţinutului câmpului.

Următoarea figură ilustrează modul în care se memorează date în câmpurile „memo", folosind ca exemplu câmpul **adresa** din tabela **ANGAJATI**.



Toate operaţiile sunt invizibile pentru utilizator, Visual FoxPro controlând în întregime mecanismul.

Încărcarea unor date într-un câmp memo se poate face fie manual de către utilizator, introducându-le caracter cu caracter într-o fereastră de editare, fie prin citirea dintr-un fişier sau din memoria calculatorului.

Prima metodă se desfăşoară în felul următor:

- se deschide o fereastră de editare pentru modificarea conţinutului tabelei, folosindu-se, de exemplu, comanda **BROWSE**;
- se poziţionează cursorul în câmpul „memo" respectiv al înregistrării dorite, după care se apasă combinaţia de taste Ctrl+PgDn. Folosind mouse-ul, se

execută un clic dublu pe câmp. Ca efect, pe ecran este deschisă o fereastră de editare în care utilizatorul are posibilitatea de a introduce caracter cu caracter conţinutul câmpului respectiv;

- după terminarea editării câmpului „memo", se iese din fereastra de editare folosindu-se combinaţia de taste Ctrl+W, combinaţie care realizează şi salvarea modificărilor în tabelă, sau prin apăsarea tastei Escape, ce nu salvează modificările aduse câmpului;
- după închiderea ferestrei de editare a câmpului „memo", se revine în fereastra **Browse**.

Metoda de încărcare a unui câmp „memo"prin cod are la bază comanda APPEND MEMO:

```
APPEND MEMO <câmp memo> FROM <fişier> OVERWRITE
```

Conţinutul întregului fişier specificat în clauza FROM se adaugă la sfârşitul câmpului „memo". Dacă se include opţiunea OVERWRITE, conţinutul fişierului nu va fi adăugat la sfârşitul câmpului „memo", ci va fi copiat peste acesta, vechiul conţinut al câmpului pierzându-se.

Exemplu

Se va adăuga o nouă înregistrare la tabela ANGAJATI, *câmpul* adresa *fiind completat din fişierul* ADRESA.TXT *ce conţine informaţiile respective.*

```
USE angajati
APPEND BLANK
REPLACE nume WITH ...
APPEND MEMO adresa FROM adresa.txt OVERWRITE
USE
```

Operaţia inversă, de copiere a conţinutului unui câmp memo dintr-o tabelă într-un fişier text (ASCII), este realizată de comanda COPY MEMO:

```
COPY MEMO <câmp memo> TO <fişier> ADDITIVE
```

Comanda va avea ca efect copierea conţinutului câmpului „memo" respectiv în fişierul precizat. Dacă fişierul nu exista anterior, se creează unul nou, altfel copierea va avea loc în fişierul existent deja pe disc. În cazul în care nu se specifică nici o extensie pentru fişier, FoxPro îi va atribui automat extensia .TXT. În mod normal, conţinutul câmpului memo va fi copiat peste conţinutul fişierului (dacă acesta există), vechiul conţinut pierzându-se. Dacă se doreşte adăugarea la sfârşitul fişierului, se include în comandă clauza ADDITIVE.

Exemplu

Se vor copia adresele primelor trei persoane din tabela **ANGAJATI** *în fişierul* **ADRESA.TXT**.

```
USE angajati
COPY MEMO adresa TO adresa.txt
NOTE se copiaza prima adresa
GOTO 2
COPY MEMO adresa TO adresa.txt ADDITIVE
NOTE se adauga a doua adresa
GOTO 3
COPY MEMO adresa TO adresa.txt ADDITIVE
NOTE se adauga a treia adresa
MODIFY FILE adresa.txt NOEDIT
NOTE se vizualizeaza fisierul adresa.txt
USE
```

Pentru încărcarea unui câmp „memo" se poate folosi ca sursă un alt câmp „memo", din aceeaşi tabelă sau din alta. Acest lucru se realizează direct cu comanda `REPLACE`.

Majoritatea funcţiilor care se aplică şirurilor de caractere funcţionează şi pentru câmpurile de tip „memo". Există însă două funcţii specifice acestui tip de câmp şi anume `MEMLINES()` şi `MLINE()`. Prima dintre ele primeşte ca parametru câmpul „memo" şi returnează numărul de rânduri ale câmpului.

```
MEMLINES(<câmp memo>)
```

Pentru a extrage un anumit rând dintr-un câmp memo se foloseşte funcţia `MLINE()`:

```
MLINE(<câmp memo>, <expresie1>, <expresie2>)
```

Primul argument reprezintă câmpul din care se extrage rândul, al doilea specifică numărul rândului, iar al treilea argument, dacă există, indică deplasamentul faţă de începutul rândului de la care începe extragerea caracterelor (se consideră câmpul „memo" ca începând de la al (`<expresie2>+1`)-lea caracter).

Exemplu

Să presupunem că în câmpul **adresa** *al înregistrării curente din tabela* **ANGAJATI** *avem următorul text:*

Bucuresti
Strada Cuza Voda, nr.54
sector 4

```
? MEMLINE (adresa)
NOTE  numarul de randuri din campul memo
3
? MLINE (adresa,2)
Strada Cuza Voda, nr.54
? MLINE (adresa,1,0)
Bucuresti
? MLINE (adresa,1,1)
ucuresti
? MLINE (adresa,1,11)
Strada Cuza Voda, nr.54
? MLINE (adresa,1,12)
trada Cuza Voda, nr.54
? MLINE (adresa,4)==""
.T.
```

Câmpurile „generale"

Câmpurile de uz general sunt folosite pentru memorarea datelor de diverse tipuri, create cu alte aplicații decât Visual FoxPro (precum documente create în Word, foi de calcul tabelar create în Excel etc.). În spatele acestor câmpuri stă tehnologia OLE (de încorporare şi legare a obiectelor), tehnologie prezentată într-un paragraf special al acestei cărţi, în capitolul „Tehnici speciale disponibile în Visual FoxPro".

Baze de date relaționale. Relații între tabele

Conform modelului relațional, o bază de date este alcătuită din mai multe tabele între care se stabilesc relații. Aceste relații pot fi permanente, memorate în fişierul bazei de date, sau temporare, construite la rularea programului şi distruse la terminarea rulării. Relațiile permanente au fost prezentate în capitolul referitor la construirea bazelor de date, iar în prezentul paragraf ne vom ocupa de construirea dinamică a relațiilor dintre tabele, adică de relațiile temporare.

Pentru prezentarea din acest paragraf vom folosi ca exemplu tabela ANGAJATI, cu structura:

1	NUME	Şir car.	14
2	PRENUME	Şir car.	30
3	COD_NUM_P	Şir car.	13
4	DATA_NAST	Dată cal.	8
5	SEXUL	Logic	1
6	ADRESA	Memo	4
7	TELEFON	Şir car.	12
8	STUDII	Şir car.	1
9	DATA_ANG	Dată cal.	8

10	FUNCTIA	Şir car.	3
11	DEPARTAM	Şir car.	3
12	SAL_BRUT	Întreg	4

Pe lângă această tabelă, vom mai defini una, numită FUNCTII, cu următoarea structură:

1	FUNCTIA	Şir car.	3
2	DENUMIRE	Şir car.	30
3	STUDII	Logic	1

în care vom depozita informaţiile referitoare la funcţiile ocupate de angajaţii din tabela ANGAJATI.

Observăm că cele două tabele au un câmp comun, şi anume functia, prin intermediul căruia se realizează corespondenţa dintre fiecare înregistrare din tabela ANGAJATI şi o înregistrare din tabela FUNCTII. Pentru a afla informaţii complete despre o persoană, trebuie citite datele referitoare la ea din ambele tabele.

Apare deci problema căutării în cele două tabele, de preferat simultan, a salariatului şi a funcţiei ocupate de acesta, problemă care îşi găseşte o rezolvare elegantă în acest paragraf. Vom stabili o relaţie între cele două tabele, astfel încât mutarea indicatorului de înregistrări pe o anumită înregistrare din tabela ANGAJATI (deci la o anumită persoană) să aibă ca efect mutarea automată a indicatorului de înregistrări din cea de-a doua tabelă, FUNCTII, pe înregistrarea corespunzătoare, cu aceeaşi valoare în câmpul functia.

După stabilirea acestei relaţii, căutarea unei persoane se va face numai în tabela ANGAJATI, în cealaltă tabelă căutarea efectuându-se în mod automat de către sistem.

Modul de lucru cu bazele de date relaţionale este următorul:

- mai întâi se deschid tabelele componente ale bazei de date relaţionale, fiecare în câte o zonă de lucru distinctă;
- urmează stabilirea relaţiilor între tabele;
- se execută operaţiile asupra bazei de date relaţionale, adică asupra tabelelor componente (adăugare de date, modificarea unor date existente, consultare etc.);
- în final, se înlătură relaţiile stabilite între tabelele bazei de date relaţionale şi se închid aceste tabele.

Pentru stabilirea unei relaţii între două tabele, acestea trebuie mai întâi deschise în două zone de lucru distincte. Relaţia stabilită între tabelele unei baze de date relaţionale nu este o relaţie de egalitate, ci una de subordonare: una dintre tabele va fi **părinte**, iar cealaltă va fi **copil**. Mutarea indicatorului de înregistrări al tabelei părinte pe o anumită înregistrare determină mutarea indicatorului de înregistrări al tabelei copil pe înregistrarea corespunzătoare, dar nu şi invers.

Înainte de stabilirea relației, pe lângă deschiderea celor două tabele mai trebuie îndeplinită o condiție, şi anume indexarea tabelei copil cu aceeaşi cheie de indexare cu cea a relației.

Stabilirea unei relații între două tabele se realizează cu ajutorul comenzii `SET RELATION`:

```
SET RELATION TO <expresie1> INTO <tabelă1>,
                <expresie2> INTO <tabelă2>...
  ADDITIVE
```

Comanda stabileşte o relație între tabela activă, considerată părinte, şi una sau mai multe tabele considerate copii, specificate prin aliasurile lor (`<tabelă1>`, `<tabelă2>`,...).

Cheile relațiilor, sau criteriile după care o înregistrare din tabela copil corespunde unei înregistrări din tabela părinte, sunt specificate prin expresiile `<expresie1>`, `<expresie2>`,... În general, aceste expresii reprezintă un câmp comun al tabelei părinte şi al celei copil şi, de asemenea, reprezintă cheia de indexare a tabelei copil. Dar o asemenea expresie poate fi şi o simplă expresie numerică, situație în care indicatorul de înregistrări din tabela copil va fi mutat pe înregistrarea cu numărul egal cu valoarea expresiei. O înregistrare a tabelei copil corespunde unei înregistrări a tabelei părinte dacă cele două au aceeaşi valoare a expresiei respective.

Relația între tabele funcționează în modul următor: mutarea indicatorului de înregistrări din tabela părinte determină evaluarea succesivă a expresiilor `<expresie1>`, `<expresie2>`,... În funcție de valorile acestor expresii, indicatorii de înregistrări ai tabelelor copil vor fi plasați pe înregistrările corespunzătoare înregistrării părinte.

În cazul în care în tabela copil nu se găseşte nici o înregistrare care să corespundă înregistrării părinte, indicatorul de înregistrări va fi poziționat la sfârşitul tabelei copil.

Clauza `ADDITIVE` face ca relațiile existente pentru tabela activă să nu fie şterse, ci la acestea să se adauge noua relație definită. Absența clauzei face ca noua relație să înlocuiască eventualele relații mai vechi ale tabelei active. Comanda `SET RELATION TO`, fără alte clauze, determină înlăturarea tuturor relațiilor tabelei active.

Exemplu

Rezolvarea problemei prezentate la începutul acestui paragraf este dată de următorul program:

```
CLOSE ALL
USE angajati
SELECT 2
USE functii
```

```
SET ORDER TO functia
SELECT 1
SET RELATION TO functia INTO 2
GOTO 1
? RECNO(1), RECNO(2)
? nume, b.functia
CLOSE ALL
```

Comanda SET RELATION face ca fiecărei înregistrări din tabela părinte să-i corespundă o înregistrare în tabela copil, saltul în cea de-a doua tabelă făcându-se automat. Dar ce se întâmplă în cazul când unei înregistrări a tabelei părinte îi corespund mai multe înregistrări ale tabelei copil? În această situație, comanda SET RELATION determină găsirea primeia dintre aceste înregistrări. Se spune că relația stabilită este de tipul *„una-la-una"*.

Pentru ca printr-o relație să poată fi găsite mai multe înregistrări ale tabelei copil ce corespund toate unei singure înregistrări a tabelei părinte, se va crea o relație de tipul *„una-la-mai-multe"*, folosindu-se comanda SET SKIP:

```
SET SKIP TO <tabelă1> ,<tabelă2> ...
```

Comanda transformă relațiile dintre tabela activă și tabelele specificate din relații *una-la-una* în relații de tipul *una-la-mai-multe*. Deci, pentru a folosi comanda SET SKIP asupra relațiilor unei tabele, acestea trebuie să fi fost create anterior, folosindu-se comanda SET RELATION.

Funcționarea relațiilor de tip una-la-mai-multe este următoarea: la poziționarea indicatorului de înregistrări din tabela părinte pe o anumită înregistrare, prin comenzile GOTO, LOCATE, SEEK etc, indicatorii de înregistrări din tabelele copil vor fi poziționați pe primele înregistrări care corespund înregistrării părinte. Făcând în continuare salturi în tabela părinte, folosind comanda SKIP (nu GOTO), indicatorul de înregistrări al acestei tabele nu se va modifica, în schimb, în tabelele copil vom fi poziționați succesiv pe următoarele înregistrări atașate înregistrării părinte. Aceasta se realizează atâta timp cât sunt parcurse toate înregistrările ce corespund înregistrării părinte.

Exemplu

De exemplu, am putea dori lista tuturor consilierilor (codul funcției este 'Con'*) angajați în cadrul unității economice. Pentru aceasta se stabilește o relație de tip* una-la-mai-multe *între tabela* FUNCTII *(părinte) și tabela* ANGAJATI *(copil). Aceasta deoarece unei înregistrări din tabela* FUNCTII *îi corespund mai multe înregistrări în tabela* ANGAJATI*.*

```
NOTE se deschid tabelele
CLOSE ALL
USE functii IN 1
```

```
SET ORDER TO functia IN 1
USE angajati IN 2
SET ORDER TO functia IN 2

NOTE  se stabileste relatia 1-1
SELECT 1
SET RELATION TO functia INTO 2  &
NOTE  se schimba tipul relatiei in 1-n
SET SKIP TO 2
NOTE  se schimba tipul relatiei in 1-n

NOTE  se parcurg inregistrarile
SELECT functii
SEEK 'Con'     && se gaseste primul consilier
SELECT 2
n=RECNO('functii')
DO WHILE n=RECNO('functii')
  NOTE  criteriul de iesire este schimbarea inregistrarii;
           curente in tabela parinte (FUNCTII)
  ? nume,prenume
  SKIP
ENDDO
CLOSE ALL
```

Înlăturarea unei relaţii dintre două tabele se face cu comanda:

```
SET RELATION OFF INTO <tabelă copil>
```

Ca urmare, se şterge relaţia dintre tabela activă şi tabela copil specificată. Spre deosebire de `SET RELATION TO`, care şterge toate relaţiile tabelei active, `SET RELATION OFF` şterge numai relaţia specificată în comandă.

Interogarea bazelor de date

Capitolul 5

- Generalități
- Crearea interogărilor bazelor de date
  - ✓ Modul de lucru cu interogările bazelor de date
  - ✓ Constructorul de interogări (**Query Designer**)
- Vederi ale bazelor de date
  - ✓ Modul de lucru cu vederile bazelor de date
  - ✓ Constructorul de vederi (**View Designer**)

Generalități

Interogarea reprezintă unul din scopurile principale ale organizării datelor în baze de date. Cu alte cuvinte, toate operațiile efectuate cu o bază de date (crearea bazei de date, încărcarea datelor, ordonarea lor etc.) au ca finalitate obținerea de diferite rapoarte (în diferite forme) pe baza datelor respective.

Def *Extragerea (fără ştergere) unor date din bazele de date, în funcție de anumite criterii şi într-un anumit format extern, se numeşte* ***interogare****.*

Termenul „interogare" poate părea nepotrivit, dar el este încetățenit în domeniul bazelor de date. De fapt, el exprimă punctul de vedere al utilizatorului, care întreabă baza de date: „*Ce date ai care respectă următoarele condiții...?*", iar aceasta îi răspunde: „*Am următoarele date care corespund condițiilor respective...*".

Pentru interogarea bazelor de date, sistemele informatice specializate (SGBD) au implementate o serie de instrumente, tehnologii şi proceduri, care permit utilizatorului specificarea celor mai diverse criterii de interogare. Printre acestea se numără şi limbajul SQL, care reprezintă de fapt un limbaj standard de interogare (şi nu numai) a bazelor de date relaționale.

SQL este implementat şi în Visual FoxPro (şi chiar şi în versiunile mai vechi), prin intermediul instrucțiunii `SELECT`. De asemenea, mai există o serie de instrucțiuni conexe (`INSERT SQL`, `DELETE SQL`, `CREATE TABLE SQL` etc.) care permit realizarea altor operații decât cele de interogare a bazelor de date, cum ar fi adăugarea de noi înregistrări, ştergerea unora existente etc.

Corespunzător comenzilor SQL de interogare a bazelor de date, în Visual FoxPro (dar şi în versiunile anterioare ale SGBD) au fost dezvoltate o serie de instrumente cu ajutorul cărora utilizatorul poate construi în mod interactiv interogări ale bazelor de date. Un prim exemplu este RQBE, un utilitar prezent în FoxPro 2.6. Corespunzător acestuia, în Visual FoxPro există „Constructorul de interogări" (**Query Designer**) şi „Constructorul de vederi" (**View Desinger**), două instrumente oarecum asemănătoare.

Def. *O* ***interogare a unei baze de date*** *reprezintă un ansamblu de specificații (tabele, câmpuri ale acestora, condiții de selecție etc.) pe baza cărora sunt extrase date din una sau mai multe baze de date.*

Interogările sunt foarte des folosite pentru construirea rapoartelor în cadrul unui sistem informatic, deoarece ele permit selectarea, pe baza diferitelor criterii, a datelor din bazele de date ale sistemului informatic respectiv.

Datele extrase printr-o interogare sunt prezentate utilizatorului în diferite formate. Altfel spus, aceste date sunt trimise spre diverse destinații, cum ar fi: ferestre de editare a tabelelor (**Browse**), grafice, rapoarte etc. Una dintre destinațiile unei operații de

interogare a unei baze de date o reprezintă tabelele, permanente sau temporare. Aceste tabele conţin de fapt submulţimi de date extrase din tabelele sursă. Modificarea acestor date nu determină însă şi modificarea automată a datelor sursă. Prin urmare, legătura dintre cele două tabele este unidirecţională.

O altă abordare a acestei probleme este dată de vederi.

Def

*O **vedere a unei baze de date** reprezintă un tip special de tabelă, construită pe baza datelor din baza de date respectivă, tabelă care permite actualizarea automată a datelor sursă în funcţie de modificările datelor extrase.*

Prin urmare, în timp ce modificarea datelor tabelelor, permanente sau temporare, create prin intermediul unei interogări, nu determină modificarea corespunzătoare a datelor din baza de date sursă, la modificarea datelor unei vederi a unei baze de date, datele din acea bază de date sunt actualizate în mod corespunzător.

Crearea interogărilor bazelor de date

Obs

În prezentul paragraf vom folosi ca exemplu baza de date ANGAJATI.DBC a angajaţilor unei unităţi economice, bază de date formată din două tabele: ANGAJATI.DBF, cu date referitoare la angajaţi, şi FUNCTII.DBF, cu date referitoare la funcţiile ocupate de aceştia. Tabelele au următoarele structuri:

ANGAJATI

Nr.crt.	Denumire	Tip	Lung.,zec.	Index	NULL
1	NUME	Şir car.	14	Nu	-
2	PRENUME	Şir car.	30	Nu	-
3	COD_NUM_P	Şir car.	13	Da	-
4	DATA_NAST	Dată cal.	8	Da	-
5	SEXUL	Logic	1	Nu	-
6	ADRESA	Memo	4	Nu	-
7	TELEFON	Şir car.	12	Nu	-
8	STUDII	Şir car.	1	Nu	-
9	DATA_ANG	Dată cal.	8	Nu	-
10	FUNCTIA	Şir car.	3	Da	-
11	DEPARTAM	Şir car.	3	Nu	-
12	SAL_BRUT	Întreg	4	Nu	-

FUNCTII

Nr.crt.	Denumire	Tip	Lung.,zec.	Index	NULL
1	COD	Şir car.	3	Da	-
2	DENUMIRE	Şir car.	40	Nu	-
3	STUDII	Logic	1	Nu	-

Cele două tabele sunt legate între ele printr-o relaţie permanentă, stabilită după câmpul `cod` *din tabela* `FUNCTII` *şi câmpul* `functia` *din tabela* `ANGAJATI`.

Modul de lucru cu interogările bazelor de date

După cum am spus mai arătat, o interogare reprezintă un ansamblu de specificaţii, pe baza cărora sistemul extrage date dintr-o bază de date. Pentru a putea fi folosită, o interogare trebuie mai întâi creată, adică trebuie construit fişierul (.QPR) în care sunt memorate specificaţiile respective. Sistemul dispune de instrumente interactive pentru crearea acestor tipuri de fişiere, deci pentru precizarea specificaţiilor interogării (dintre care unul este Constructorul de interogări – **Query Designer**).

Ori de câte ori se doreşte extragerea datelor din baza de date, interogarea trebuie rulată. Cu alte cuvinte, la comanda utilizatorului, sistemul citeşte parametrii interogării (din fişierul respectiv) şi extrage datele pe baza acestora.

Comanda de rulare a unei interogări este DO, cu sintaxa:

```
DO <nume interogare>.qpr
```

De fapt, interogarea reprezintă un fişier ASCII, conţinând o comandă SELECT SQL. Rularea fişierului respectiv înseamnă, de fapt, execuţia comenzii respective. Este necesară precizarea explicită a extensiei fişierului, .QPR, deoarece DO presupune o extensie implicită .PRG.

Interogarea permite crearea rapoartelor. Ori de câte ori dorim prezentarea unei anumite situaţii pe baza datelor din una sau mai multe baze de date, executăm interogarea corespunzătoare (care a fost construită anterior, la proiectarea sistemului informatic), iar datele sunt furnizate la destinaţia specificată (pe ecran, la imprimantă etc.).

Există două metode de creare a interogărilor:

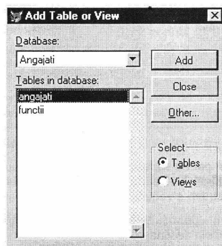
- una interactivă, instrumentul folosit fiind în acest caz Constructorul de interogări;

- una manuală, reprezentată de limbajul SQL, implementat şi în Visual FoxPro (comanda de interogare este SELECT SQL, care se poate introduce oriunde într-un program de prelucrare).

Constructorul de interogări (Query Designer)

Pornirea Constructorului de interogări în vederea construirii unei interogări noi debutează cu alegerea opţiunii **New** a meniului **File**. Urmează apoi selectarea butonului **Query** (interogare) din fereastra deschisă pe ecran şi acţionarea butonului **New file** (fişier nou).

Prima operaţie care trebuie realizată este selectarea tabelelor din care se vor extrage datele. Dacă anterior pornirii constructorului a fost deschisă o bază de date, atunci sistemul va afişa pe ecran o fereastră din care se pot selecta ca sursă de date pentru interogare (adică tabele din care se vor extrage datele) tabelele bazei de date respective.



Tabelele sursă se aleg (prin selectare şi apăsarea butonului **Add**) din lista **Tables in database** (tabele în baza de date).

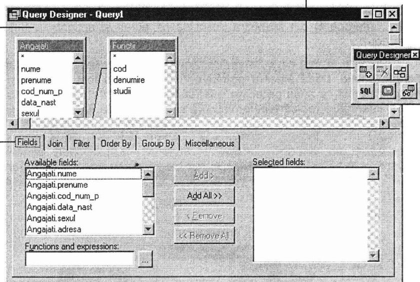
Dacă, anterior, în baza de date au fost stabilite şi legături permanente între tabele, acestea se vor păstra ca atare şi în fereastra Constructorului de interogări. În caz contrar, la adăugarea unei tabele este afişată o fereastră pentru specificarea eventualelor legături între tabele.

După specificarea surselor de date este pornit efectiv Constructorul de interogări, a cărui fereastră arată ca în figura de mai jos:

Aceasta este bara utilitară a Constructorului de interogări

În această zonă vor fi afişate tabelele sursă

Aceste butoane permit selectarea paginilor cu diferite opţiuni



Fereastra conţine în partea superioară o zonă în care sunt afişate tabelele sursă, împreună cu legăturile dintre ele, iar în partea de jos o serie de pagini în care se vor preciza parametrii de construire a vederii respective. Paginile au următoarele utilizări:

- **Fields** (câmpuri) – permite specificarea câmpurilor din tabelele sursă care vor fi selectate în tabela destinaţie;
- **Join** (legătură) – foloseşte la precizarea legăturilor stabilite între tabelele sursă;
- **Filter** (filtru) – oferă posibilitatea specificării unor condiţii de filtrare, adică de selecţie după anumite criterii a datelor din tabelele sursă;
- **Order By** (ordonare) – specifică ordinea în care vor fi prezentate datele în tabela destinaţie;

- **Group By** (grupare) – permite precizarea unor criterii de grupare a datelor, însoţite eventual de criterii de selecţie la nivel de grup (pentru alegerea anumitor grupuri în tabela destinaţie);
- **Miscellaneous** (altele) – precizează alte opţiuni, care nu au fost incluse în celelalte pagini.

Selectarea câmpurilor în noua vedere

Dintre câmpurile tabelelor sursă, unele pot fi preluate în tabela destinaţie. La limită, pot fi preluate toate aceste câmpuri, dar, pentru economie de resurse, este de preferat să fie preluate numai acelea care sunt efectiv necesare.

Exemplu

De exemplu, dacă dorim lista angajaţilor unităţii economice, cu nume, prenume, funcţie şi salariu brut, câmpurile selectate în interogare vor fi `Angajati.nume`*,* `Angajati.prenume`*,* `Functii.denumire` *şi* `Angajati.sal_brut`*.*

Pentru specificarea unui câmp care să fie preluat în noua vedere, mai întâi se activează pagina **Fields**, apoi se selectează câmpul dorit din lista **Available Fields** (câmpuri disponibile) şi se acţionează butonul **Add** (adăugare). Câmpul respectiv va fi copiat în lista **Selected Fields** (câmpuri selectate).

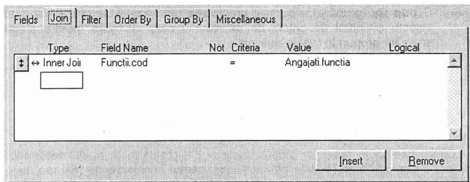
Dacă în noua vedere se doreşte introducerea unui câmp calculat, formula de calcul a acestuia se va încărca în câmpul de editare **Functions and expresions** (funcţii şi expresii), fie manual, fie cu ajutorul Constructorului de expresii, pornit la acţionarea butonului din dreapta acestui câmp de editare.

Specificarea legăturilor între tabelele şi vederile sursă

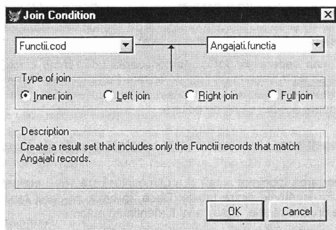
După precizarea tabelelor sursă şi a câmpurilor care vor fi preluate din acestea, trebuie specificate legăturile dintre tabele. Dacă, în baza de date, tabelele respective erau deja legate printr-o relaţie, aceasta va fi menţinută automat.

Dacă există însă tabele care nu au fost anterior în baza de date sau care nu au fost legate de o altă tabelă, există posibilitatea precizării explicite a legăturii ce urmează să lege tabelele respective cu restul.

Pentru stabilirea unei relaţii noi între două tabele, mai întâi se alege pagina **Join** a ferestrei Constructorului de vederi. Pagina respectivă arată ca în figura următoare:



Adăugarea unei condiţii goale (ce urmează a fi completată ulterior) se face prin acţionarea butonului **Insert** (inserare). Dacă legătura a fost adăugată automat de sistem (ea a fost definită în baza de date), specificarea parametrilor relaţiei se face în fereastra **Join Condition** (condiţia de legătură), deschisă la acţionarea butonului ↔ din partea stângă a liniei legăturii.



În această fereastră se pot specifica cele două câmpuri care vor face obiectul relaţiei (câmpul din tabela părinte în stânga şi cel din tabela copil în dreapta) şi, de asemenea, tipul de legătură, după cum urmează:

- **Inner join** (legătură interioară) – în tabela destinaţie vor fi încărcate doar înregistrările care respectă condiţia impusă de relaţie, deci care au

corespondent în tabela pereche (înregistrările din tabela părinte care au corespondent în tabela copil şi invers);

- **Left join** (legătură la stânga) – în tabela destinaţie vor fi incluse doar înregistrările care respectă condiţia relaţiei (au corespondent în tabela vecină). În plus, în tabela destinaţie vor fi incluse şi înregistrările din tabela părinte (din stânga) care nu au corespondent în tabela copil (din dreapta);
- **Right join** (legătură la dreapta) – în tabela destinaţie vor fi incluse doar înregistrările care respectă condiţia relaţiei. La acestea se adaugă şi înregistrările din tabela copil (din dreapta) care nu au corespondent în tabela părinte (din stânga);
- **Full join** (legătură completă) – în tabela destinaţie vor fi incluse toate înregistrările;

Exemplu

Pentru relaţia funcţie→angajat, în care tabela FUNCTII *este părinte (în stânga), iar tabela* ANGAJATI *este copil (în dreapta), putem avea următoarele cazuri:*

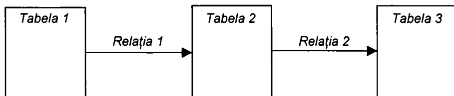
- *dacă dorim afişarea angajaţilor cu denumirile complete ale funcţiilor lor, dar vrem să nu fie afişate persoanele ale căror funcţii nu sunt descrise în tabela de funcţii, vom folosi legătura interioară (**Inner join**);*
- *pentru afişarea funcţiilor care sunt ocupate în unitatea respectivă (există cel puţin un angajat cu acea funcţie) putem folosi relaţia la stânga (**Left join**).*

În cazul în care legătura nu a fost definită anterior în baza de date, atunci ea trebuie introdusă manual în pagina **Join** a ferestrei Constructorului de interogări. În câmpul **Type** (tip) se introduce tipul legăturii (a se vedea mai sus), în **Field Name** (nume câmp) se introduce câmpul părinte sau, în cazul unei expresii, partea stângă a condiţiei relaţiei, iar în câmpul **Value** (valoare) se introduce câmpul copil sau partea dreaptă a condiţiei relaţiei. Cele două părţi ale condiţiei sunt legate printr-un operator introdus în câmpul **Criteria** (criterii). Prin acţionarea butonului din coloana **Not** (nu), se obţine negarea condiţiei respective.

În fine, ultima coloană a listei se numeşte **Logical** (logic) şi permite specificarea unui operator logic binar (**AND** – *Şi logic* – sau **OR** – *Sau logic*), care leagă între ele condiţiile respective (în cazul în care sunt mai multe).

Exemplu

Să luăm exemplul a trei tabele, legate între ele prin următoarele relaţii:



*În acest caz sunt necesare două legături, unite între ele prin operatorul **AND**. Prima dintre legături se stabileşte între prima tabelă şi cea de-a doua, iar a două legătură între tabela a doua şi tabela a treia.*

Este, de asemenea, importantă şi ordinea celor două legături. Evaluarea celor două legături se face astfel: mai întâi se stabileşte prima legătură specificată în listă, rezultând un grup de înregistrări. Acest grup este folosit ulterior ca sursă în evaluarea celei de-a doua legături, în urma căreia rezultă un al doilea grup de înregistrări şi aşa mai departe.

Condiţiile de selecţie a înregistrărilor

Una dintre principalele funcţiuni ale unei interogări este aceea de selecţie. Cu alte cuvinte, cu ajutorul unei interogări se extrag din baza de date numai anumite date. Pentru aceasta se folosesc diferite criterii de selecţie, criterii care sunt precizate în pagina **Filter** (filtru) a Constructorului de interogări.

Fields | Join | Filter | Order By | Group By | Miscellaneous

Field Name	Not	Criteria	Example	Case	Logical
Angajati.sexul		=	.T.		<None>

Insert | Remove

Condiția de selecție a înregistrărilor are următoarea structură:

```
<câmp sau expresie> <operator> <valoare>
```

Cu alte cuvinte, pentru fiecare înregistrare se testează dacă valoarea din `<câmp>` este în relația dată de `<operator>` (egalitate, mai mare,...) cu `<valoare>`. În caz afirmativ, înregistrarea va fi inclusă în tabela destinație, altfel ea va fi ignorată.

`<câmp>` se introduce în **Field Name** (nume câmp), `<operator>` în **Criteria** (criterii), iar `<valoare>` în **Example** (exemplu). Dacă se dorește negarea condiției, se activează comutatorul **NOT**, iar în cazul în care `<câmp>` este de tip șir de caractere se poate folosi comutatorul **Case**, care, în starea activat, indică o testare dependentă de tipul literelor, mari sau mici.

Exemplu

De exemplu, în figura prezentată este precizată condiția:

```
Angajati.sexul=.T.
```

adică se vor selecta doar angajații de sex masculin.

Alte selecții ar putea fi:

- *doar angajații cu o anumită funcție, să zicem secretară:*

```
Angajati.functie="Sec"
```

- *doar angajații cu salariu mai mare decât o anumită sumă (1000000):*

```
Angajati.sal_brut>=1000000
```

- *angajații bărbați, cu studii superioare, angajați începând cu data de 3 ianuarie 1990:*

```
Angajati.sexul=.T. AND Angajati.studii='S' AND;
      Angajati.data_ang>={01/03/1990}
```

Pot fi specificate mai multe condiții de selecție, legate între ele prin operatorii logici **AND** sau **OR**. Dacă se dorește selectarea unor înregistrări care respectă simultan două condiții, acestea se vor lega prin operatorul **AND** (Și logic). Dacă însă se dorește selectarea unor înregistrări care respectă cel puțin una dintre condiții, atunci condițiile vor fi legate prin operatorul **OR** (Sau logic).

Exemplu

Dacă se dorește selectarea angajaților bărbați, cu studii superioare, angajați după data de 3 ianuarie 1990, se va folosi condiția:

```
Angajati.sexul=.T. AND Angajati.studii='S' AND;
    Angajati.data_ang>={01/03/1990}
```

În cazul mai multor condiții, legate între ele prin unul dintre cei doi operatori, ordinea de evaluare este următoarea: mai întâi se evaluează condițiile legate prin **AND** şi apoi cele legate prin **OR**.

Exemplu

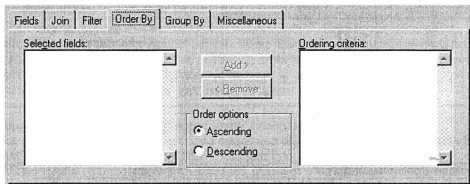
Următoarele două secvențe sunt echivalente:

```
<condiţie 1> OR <condiţie 2> AND <condiţie 3>
<condiţie 1> OR (<condiţie 2> AND <condiţie 3>)
```

Ordonarea rezultatelor

Rezultatele interogării pot fi furnizate într-o anumită ordine, dată de anumite criterii. De exemplu, lista angajaților unității economice este utilă în ordinea alfabetică a numelor şi a prenumelor acestora sau în ordinea crescătoare a salariilor.

Pentru specificarea ordinii în care vor fi furnizate datele de către interogare se foloseşte pagina **Order By** (ordonate după):



Pentru a stabilii că ordinea înregistrărilor în tabela destinație va fi dată de un anumit câmp, acest câmp trebuie selectat în lista din partea stângă a paginii şi copiat în cea din partea dreaptă (operație realizată cu ajutorul butonului **Add**). Se pot stabili mai multe criterii de ordonare, prin selectarea mai multor câmpuri în lista din dreapta a paginii **Order By**. Un criteriu de ordonare este luat în considerare numai în cazul în

care, după aplicarea criteriilor anterioare, nu s-a putut stabili ordinea unor înregistrări (acestea având aceleaşi valori în câmpurile care dau ordinea înregistrărilor).

Exemplu

De exemplu, dacă doi angajaţi au acelaşi nume de familie, criteriul următor va fi cel al prenumelui. Pentru aceasta, trebuie selectate pe post de câmpuri de ordonare câmpurile `Angajati.nume` *şi* `Angajati.prenume`*, în această ordine.*

Gruparea datelor

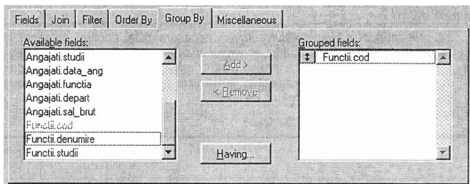
Gruparea reprezintă o altă facilitate oferită de Visual FoxPro la realizarea interogărilor unei baze de date. În urma grupării, una sau mai multe înregistrări sunt cumulate după anumite criterii, pentru fiecare grup de înregistrări fiind furnizată în tabela destinaţie o singură înregistrare.

Gruparea se realizează după unul sau mai multe câmpuri ale tabelei, care vor alcătui astfel cheia de grupare. Înregistrările care vor avea aceeaşi valoare a cheii de grupare (adică aceleaşi valori în câmpurile componente ale cheii) vor forma un singur grup, care va da în tabela destinaţie o singură înregistrare. Câmpurile numerice ale acestei înregistrări vor fi calculate prin însumarea câmpurilor respective din toate înregistrările aceluiaşi grup.

Exemplu

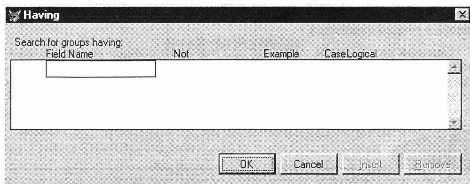
De exemplu, pentru tabela angajaţilor s-ar putea solicita sumele necesare plăţii fiecărei categorii de angajaţi, directori, consilieri etc. Pentru aceasta, se va realiza o grupare după funcţiile angajaţilor, în câmpul salariului obţinându-se în final salariul cumulat pentru fiecare funcţie în parte.

Pagina corespunzătoare grupării în Constructorul de interogări este **Group By** (grupare după).



Această pagină conţine două liste, cea din stânga cuprinzând câmpurile după care s-ar putea realiza gruparea, iar cea din dreapta câmpurile care intră efectiv în cheia de grupare. Sarcina proiectantului interogării este selectarea în lista din stânga a câmpurilor care vor trece în lista din dreapta (cu butonul **Add**). Câmpurile selectate vor alcătui împreună cheia de grupare, în sensul că mai multe înregistrări vor fi considerate un grup numai dacă valorile din câmpurile selectate vor fi aceleaşi.

Constructorul de interogări oferă proiectantului posibilitatea selectării grupurilor de înregistrări. Cu alte cuvinte, putem impune o condiţie care trebuie respectată de grupurile de înregistrări pentru a fi preluate în tabela finală. Această condiţie se specifică în fereastra deschisă ca urmare a acţionării butonului **Having** din partea inferioară a paginii **Group By** din Constructorul de interogări.



Expresia se construieşte asemănător cu cea specificată pentru selectarea înregistrărilor, adică în formatul:

```
<câmp sau expresie> <operator> <valoare>
```

Exemplu

Să presupunem că din totalul sumelor necesare plăţii fiecărei categorii de personal (raportul prezentat în exemplul anterior) nu ne interesează decât suma corespunzătoare secretarelor. Pentru aceasta, se realizează o grupare după funcţii, iar apoi se limitează afişarea doar la acel grup de angajaţi care sunt secretare. Condiţia este una de tipul:

```
Angajati.functie="Sec"
```

Destinația interogării

O interogare realizează extragerea unor date din tabelele sursă, date pe care le trimite la o anumită destinație. Aceasta poate fi:

- o fereastră de editare **Browse**;
- o tabelă permanentă (**Table**) sau temporară (**Cursor**). Aceasta poate fi preluată ulterior pentru o nouă prelucrare sau pentru a fi prezentată utilizatorului într-un format extern (un raport);
- un grafic (**Graph**). Se poate folosi ca destinație un grafic construit cu o aplicație specială a sistemului, numită Microsoft Graph;
- ecranul (**Screen**), adică fereastra principală a sistemului Visual FoxPro;
- un raport (**Report**) construit cu ajutorul Constructorului de rapoarte;
- un set de etichete (**Label**).

Selectarea se realizează prin acționarea butonului de pe bara utilitară a Constructorului de interogări, urmată de selectarea destinației dorite din fereastra deschisă pe ecran.

Instrucțiunea SQL echivalentă cu o interogare

Una dintre facilitățile importante ale Constructorului de interogări este posibilitatea aflării în orice moment a instrucțiunii SQL SELECT echivalentă cu interogarea în curs de proiectare. Aceasta permite proiectantului să învețe limbajul SQL în mod interactiv, ceea ce este foarte util, dacă avem în vedere standardul reprezentat de acest limbaj.

De asemenea, se poate edita fișierul interogării, acesta fiind unul de tip text, care conține instrucțiunea SELECT respectivă. În felul acesta, se poate folosi Constructorul de interogări în paralel cu metoda manuală, ceea ce oferă o flexibilitate sporită.

Pentru a vedea la un moment dat instrucțiunea SQL echivalentă cu interogarea în curs de construire, se apasă butonul SQL de pe bara utilitară a Constructorului de interogări. Pe ecran este deschisă o fereastră în care este afișată instrucțiunea respectivă (care însă nu se poate modifica manual).

Rularea interogării

Pentru a furniza rezultatele dorite de utilizator, o interogare trebuie rulată, aceasta însemnând de fapt execuția comenzii SELECT respective. O interogare poate fi rulată direct din Constructorul de interogări, prin alegerea opțiunii **Run Query** (rulare interogare) a meniului **Query**.

Rularea prin cod a interogării se face cu ajutorul comenzii DO:

```
DO <interogare>.qpr
```

Parametrii unei interogări

Există situații în care este necesară rularea unei interogări cu nişte parametri din exterior. Acesta este, de exemplu, cazul unei interogări folosite într-un program de raportare. În general, un astfel de program constă dintr-o formă în care utilizatorul îşi precizează opțiunile referitoare la raport (ce doreşte). Din formă se apelează una sau mai multe interogări, cu parametrii furnizați de formă în funcție de selecțiile utilizatorului. Aceste interogări urmează să extragă datele solicitate de utilizator, care apoi sunt preluate într-un raport afişat pe ecran sau la imprimantă.

De exemplu, pentru selecția angajaților unității economice în funcție de studii, în forma programului de raportare trebuie prevăzută o listă prin care utilizatorul să-şi precizeze alegerea (ce studii trebuie să aibă angajații pentru a fi selectați în raport).

Pentru a putea realiza o interogare cu parametri, trebuie folosită metoda manuală, adică editarea fişierului interogării respective. Acest fişier conține, de fapt, instrucțiunea SELECT corespunzătoare:

```
<instrucţiune SELECT>
```

Se modifică acest fişier astfel:

```
PARAMETERS <listă de parametri locali>
  <instrucţiune SELECT cu parametri>
```

Cu alte cuvinte, înaintea instrucțiunii SELECT se introduce linia PARAMETERS, prin care se realizează transferul parametrilor. Apoi, în instrucțiunea SELECT propriu-zisă se înlocuiesc constantele cu parametrii locali corespunzători.

Exemplu

Să presupunem că dorim extragerea angajaților ale căror studii sunt precizate din exterior ('S' pentru studii superioare, 'M' pentru studii medii şi 'F' pentru cazul fără studii). Comanda SELECT construită cu ajutorul Constructorului de interogări este de forma:

```
SELECT ...
  WHERE Angajati.studii=="S";
  ...
```

Această comandă se transformă în:

```
PARAMETERS studii_1
  SELECT ...
    WHERE Angajati.studii==studii_1;
    ...
```

Rularea unei interogări cu parametri se face printr-o comandă de tipul:

```
DO <nume interogare>.qpr WITH <listă parametri>
```

Exemplu

Interogarea prezentată în exemplul anterior se apelează printr-o instrucţiune de tipul:

```
DO ang_st.qpr WITH 'S'
```

sau

```
st='S'
DO ang_st.qpr WITH st
```

Dezavantajul folosirii metodei de mai sus, adică editarea manuală a fişierului interogării, este faptul că interogarea nu mai poate fi editată ulterior cu ajutorul Constructorului de interogări, deoarece acesta nu cunoaşte instrucţiunea PARAMETERS (cunoaşte formatul standard).

Vederi ale bazelor de date

Modul de lucru cu vederile bazelor de date

În capitolul referitor la construirea bazelor de date am arăta că:

Def

vederile reprezintă un tip special de tabele, construite pe baza datelor din una sau mai multe tabele sau vederi, legate între ele prin relații. Ele reprezintă un alt „punct de ***vedere****" asupra datelor dintr-o bază de date.*

Construirea unei vederi constă, de fapt, în extragerea pe baza unor criterii a unui set de date dintr-o bază de date. De aceea, vederile sunt asemănătoare interogărilor (şi acestea extrag date din bazele de date). Diferența dintre o vedere şi o interogare este faptul că prima dintre ele, vederea, reprezintă o structură logică ce conține date, pe când interogarea reprezintă programul prin care se construieşte tabela rezultat, permanentă sau temporară.

O vedere este memorată în fişierul bazei de date (.DBC) şi, de aceea, folosirea ei trebuie precedată de deschiderea bazei de date respective. Imediat după ce această condiție este îndeplinită, vederea se poate folosi ca orice altă tabelă a bazei de date. O interogare este însă memorată într-un fişier ASCII independent, care trebuie rulat pentru a se obține datele respective.

Interogările sunt folosite pentru extragerea datelor din una sau mai multe tabele sursă şi prezentarea lor în exterior într-un anumit format. Prin urmare, fluxul datelor este unidirecțional (de la tabelele sursă la tabela destinație). Modificarea unor date în tabela creată de o interogare nu are nici un efect asupra datelor din tabelele de unde au fost preluate datele respective. În schimb, dacă se modifică unele date într-o vedere, în funcție de opțiunea utilizatorului, schimbările respective se pot transmite şi tabelelor sursă. Prin urmare, în cazul vederilor, fluxul de date dintre tabelele sursă şi vedere poate fi bidirecțional.

Proiectarea vederilor este foarte asemănătoare cu proiectarea interogărilor şi, de aceea, cele două utilitare, Constructorul de vederi şi cel de interogări, diferă foarte puțin. De fapt, diferențele constau în acele opțiuni referitoare la actualizarea datelor sursă, care sunt prezente numai în cazul Constructorului de vederi.

Datorită faptului că într-un paragraf anterior a fost prezentat pe larg Constructorul de interogări, în cele ce urmează nu vor fi tratate decât aspectele specifice utilitarului folosit în cazul vederilor.

Constructorul de vederi (View Designer)

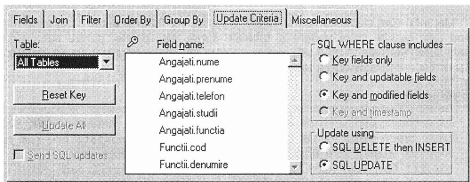
Vederile sunt asociate unei baze de date şi, prin urmare, ele sunt memorate direct în fişierul bazei de date respective (cu extensia .DBC). Deci, înainte de a proceda la construirea unei interogări, trebuie deschisă baza de date. Pentru aceasta, se poate introduce în fereastra de comenzi instrucţiunea:

```
OPEN DATABASE <nume bază de date>
```

Pentru adăugarea la baza de date a unei vederi, se alege opţiunea **New** a meniului **File**. În fereastra deschisă pe ecran se alege butonul **View** (vedere) şi apoi se apasă butonul **New file** (fişier nou). Ca urmare a acestor operaţii este pornit Constructorul de vederi.

Lucrul cu acest utilitar începe cu precizarea tabelelor sursă şi a legăturilor între ele. Urmează specificarea câmpurilor care vor fi preluate în noua vedere (pagina **Fields**), a condiţiilor de selecţie a înregistrărilor (pagina **Filter**), a criteriilor de ordonare (pagina **Order By**) şi a celor de grupare (pagina **Group By**).

Opţiunile referitoare la actualizarea datelor sursă pe baza modificărilor datelor din vedere se precizează într-o pagină distinctă a Constructorului de vederi, pagină numită **Update Criteria** (criterii de actualizare).



Din punct de vedere al mecanismului de realizare a corespondenţei vedere – tabele sursă, avem două tipuri de câmpuri:

- *câmpuri cheie*, care sunt acele câmpuri după care se stabileşte corespondenţa între vedere şi tabelele sursă;
- *câmpuri noncheie*, care nu sunt folosite la realizarea acestei corespondenţe.

Din punct de vedere al transferului modificărilor din vedere către tabelele sursă, avem două feluri de câmpuri:

- *câmpuri actualizate*, în sensul că orice modificare a lor în vedere este transmisă tabelelor sursă;
- *câmpuri neactualizate*, adică acele câmpuri care, chiar dacă sunt modificate în vedere, nu sunt afectate în tabelele sursă.

În ceea ce priveşte actualizarea tabelelor sursă, trebuie spus că nu toate câmpurile acestora care sunt preluate în vedere pot fi actualizate. Un exemplu în acest sens îl constituie câmpurile calculate, care nu pot fi actualizate în vedere, deoarece corespondenţa nu este biunivocă.

Construirea unei vederi presupune specificarea atât a câmpurilor cheie, cât şi a câmpurilor actualizate. Lista derulantă **Table** a paginii **Update Criteria** a Constructorului de vederi este folosită pentru specificarea tabelelor care vor da câmpuri ce vor face obiectul actualizării. Alegând opţiunea **All Tables** (toate tabelele), în lista câmpurilor disponibile pentru actualizare (**Field name** – nume câmp) vor fi afişate numele câmpurilor tuturor tabelelor implicate în vedere. Prin urmare, specificarea opţiunilor referitoare la actualizare începe cu alegerea opţiunii dorite din această listă (fie toate tabelele, fie numai unele dintre acestea).

Lista **Field name** (nume câmp) a paginii conţine toate câmpurile care sunt disponibile pentru actualizare. Urmează acum stabilirea pentru fiecare câmp a tipului său referitor la actualizare, adică dacă este câmp cheie sau câmp de actualizat.

Mai întâi vom stabili câmpurile cheie. Acestea vor fi alese astfel încât să identifice în mod unic relaţia dintre vedere şi tabelele sursă. Mai precis, pentru fiecare dintre tabelele sursă vom alege unul sau mai multe câmpuri cheie, care trebuie să identifice în mod unic fiecare înregistrare a tabelei respective.

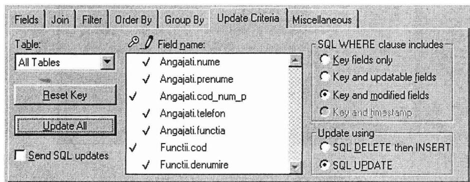
Exemplu

De exemplu, pentru tabela angajaţilor unităţii economice, `ANGAJATI.DBF`, vom folosi drept câmp cheie codul numeric personal (`Angajati.cod_num_p`), deoarece acesta este unic pentru fiecare persoană. Pentru tabela funcţiilor, `FUNCTII.DBF`, vom folosi câmpul cheie al codului (`Functii.cod`), ştiind că fiecare funcţie are un cod unic.

Pentru a stabili că un câmp este câmp cheie, se activează comutatorul asociat câmpului respectiv în coloana cheii (🔑) din lista **Field name**. În acea coloană va apărea un semn de tipul ✓ pentru a indica această stare a câmpului.

O dată indicate câmpurile cheie, se poate trece la stabilirea câmpurilor marcate pentru actualizare. Pentru aceasta se poate acţiona direct butonul **Update All**

(actualizează toate), astfel ca toate câmpurile care nu sunt câmpuri cheie să fie marcate automat pentru actualizare.



Dacă numai o parte dintre câmpuri se doresc a fi actualizate, acestea trebuie marcate manual, prin acţionarea cu mouse-ul pe comutatoarele din coloana a doua a listei ( ).

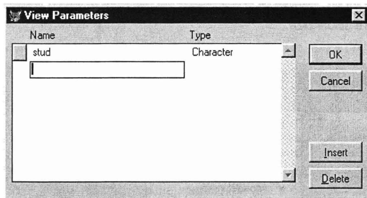
Pentru a se realiza efectiv actualizarea, trebuie activat comutatorul **Send SQL updates** (transmite actualizările SQL); în caz contrar, actualizarea nu se realizează. Modificarea câmpurilor marcate pentru actualizare se poate face în două moduri:

- prin ştergerea înregistrărilor care au suferit modificări şi reinserarea lor în varianta nouă, modificată (cazul selectării butonului **SQL DELETE then INSERT**);
- prin scrierea directă în înregistrările existente (cazul selectării butonului **SQL UPDATE**).

Alte opţiuni ale paginii referitoare la actualizare sunt folosite pentru specificarea modului în care sunt tratate eventualele conflicte generate de folosirea în comun (multiutilizator) a tabelelor sursă. Nu vom trata aici acest subiect.

Parametrii unei vederi

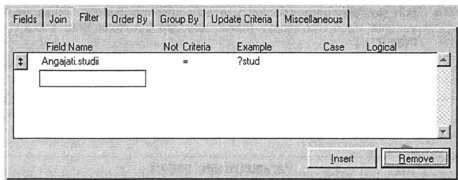
O facilitate specială a vederilor este posibilitatea de parametrizare cu valori solicitate în mod dinamic, la rulare, de la utilizator. Acest lucru se realizează prin introducerea în faţa variabilei folosite pentru memorarea valorii respective a unui semn de întrebare, ?. Variabilele utilizate ca parametri trebuie specificate la construirea vederii. Fereastra de dialog ce serveşte acestui scop este similară celei din figura următoare:



Această fereastră se deschide la alegerea opţiunii **View Parameters** (parametrii vederii) a submeniului **Query**.

Exemplu

De exemplu, din tabela **ANGAJATI** *dorim extragerea angajaţilor ale căror studii sunt precizate din exterior, în mod dinamic, la rularea interogării (*'S' *pentru studii superioare,* 'M' *pentru studii medii şi* 'F' *pentru cei fără studii). Pentru aceasta, se introduce criteriul de filtrare sub forma:*



La rulare, înainte de construirea vederii, pe ecran va fi afişată o fereastră de dialog în care utilizatorul poate introduce o valoare. Această valoare va fi încărcată în variabila **stud** *şi va fi folosită pentru filtrarea datelor (se vor alege acei angajaţi care au studiile specificate).*

Partea a III-a – PROGRAME

Elemente de programare clasică (structurată)

Capitolul 6

- Structuri de control fundamentale
  - ✓ Structura liniară
  - ✓ Structuri ramificate
  - ✓ Structuri repetitive
- Proceduri şi funcţii definite de utilizator
  - ✓ Definirea şi apelarea modulelor
  - ✓ Variabile locale şi variabile globale
  - ✓ Transferul parametrilor la şi de la module de program

Structuri de control fundamentale

În general, comenzile componente ale unui program sunt executate în ordinea în care apar în program. Însă această ordine nu este totdeauna potrivită. În practică apar ocazii în care un grup de instrucţiuni trebuie executat numai în anumite condiţii sau trebuie executat în mod repetat. Prin urmare, a fost nevoie ca în limbajele de programare să fie introduse şi instrucţiuni care să controleze astfel de situaţii, acestea fiind prezentate în continuare.

În cadrul modelului programării structurate, există trei structuri fundamentale cu ajutorul cărora pot fi rezolvate toate situaţiile din practică. Acestea sunt: structura liniară, structura ramificată şi cea repetitivă.

Structura liniară

Aceasta este structura obişnuită, în care instrucţiunile sunt executate una după alta, în ordinea în care apar în program.

Structuri ramificate

Aceste structuri sunt cele care condiţionează executarea unui grup de instrucţiuni de îndeplinirea unei anumite condiţii şi sunt implementate prin comenzile `IF...ENDIF`, `DO CASE...ENDCASE` şi prin funcţia `IIF()`.

Comanda `IF...ENDIF` are două forme, care permit:

- executarea unor instrucţiuni numai dacă este respectată o condiţie dată;
- executarea fie a unui grup de instrucţiuni, fie a altui grup de instrucţiuni, în funcţie de rezultatul evaluării unei expresii logice (condiţii).

Prima variantă, numită „cu o singură ramură", are următoarea formă:

```
IF <condiţie>
  <grup de instrucţiuni>
ENDIF
```

Grupul de instrucţiuni încadrat de `IF` şi `ENDIF` va fi executat numai dacă este îndeplinită condiţia de după `IF`. În caz contrar, instrucţiunile sunt omise, execuţia continuând cu prima instrucţiune de după `ENDIF`.

Exemplu

În următorul exemplu, dacă valoarea variabilei a *este 0, va fi afişat un mesaj de eroare.*

```
IF a=0
  ? 'Nu se poate realiza impartirea la 0'
ENDIF
```

Cea de-a doua formă a comenzii IF, numită şi „cu două ramuri", arată astfel:

```
IF <condiţie>
  <grup 1 de instrucţiuni>
ELSE
  <grup 2 de instrucţiuni>
ENDIF
```

Execuţia începe prin evaluarea condiţiei. Dacă aceasta este îndeplinită (valoarea *adevărat*) va fi executat primul grup de instrucţiuni, iar în caz contrar (valoarea *fals*) va fi executat cel de-al doilea grup de instrucţiuni.

Această a doua formă este echivalentă cu:

```
IF <condiţie>
  <grup 1 de instrucţiuni>
ENDIF
IF NOT(<condiţie>)
  <grup 2 de instrucţiuni>
ENDIF
```

Exemplu

În următorul exemplu se compară valorile a două variabile, a *şi* b*:*

```
IF a>b
  ? 'Prima variabilă este mai mare'
ELSE
  ? 'A doua variabilă este mai mare sau cele două sunt egale'
ENDIF
```

Comenzile IF pot fi incluse una în alta, realizându-se astfel structuri complexe.

Exemplu

De exemplu, compararea celor două variabile ar putea arăta astfel:

```
IF a>b
  ? 'Prima variabilă este mai mare'
ELSE
  IF a=b
    ? 'Cele două varibile sunt egale'
  ELSE
    ? 'A doua variabilă este mai mare'
  ENDIF
ENDIF
```

Un efect asemănător cu cel al comenzii `IF...ENDIF`, de selecţie dintre două variante, se obţine prin funcţia `IIF()`:

```
IIF (<condiţie>, <valoare1>, <valoare2>)
```

Funcţia evaluează condiţia şi returnează `<valoare1>` când aceasta este îndeplinită şi `<valoare2>` în caz contrar. Cele două valori nu trebuie neapărat să fie de acelaşi tip.

Exemplu

Compararea a două varibile se poate realiza cu funcţia `IIF()`:

```
? IIF(a>b,'Prima variabilă este mai mare',;
          'A doua variabilă este mai mare sau ele sunt egale')
```

sau

```
? IIF(a>b,'Prima variabilă este mai mare',;
       IIF(a=b,'Variabilele sunt egale',;
                 'A doua variabilă este mai mare'))
```

În această a doua variantă, funcţia `IIF()` a fost folosită de două ori, a doua oară în interiorul primeia.

Un alt tip de structură ramificată este instrucţiunea `DO CASE...ENDCASE`, având următoarea sintaxă:

```
DO CASE
  CASE <condiţie1>
    <grup 1 de instrucţiuni>
  CASE <condiţie2>
    <grup 2 de instrucţiuni>
  ...
```

```
OTHERWISE
   <grup n de instrucţiuni>
ENDCASE
```

Comanda va determina execuţia grupului de instrucţiuni pentru care este îndeplinită condiţia specificată în linia CASE (valoarea *adevărat*). Execuţia comenzii decurge în modul următor: se evaluează prima condiţie şi, dacă valoarea obţinută este *adevărat*, se execută grupul 1 de instrucţiuni. Dacă expresia logică respectivă are valoarea *fals*, se trece la evaluarea următoarei expresii logice. După găsirea primei condiţii îndeplinite şi executarea grupului de instrucţiuni corespunzător, execuţia comenzii se încheie, programul continuând cu prima comandă de după ENDCASE.

Dacă nici una dintre condiţiile specificate nu este îndeplinită, apar două cazuri:

- când nu există clauza OTHERWISE, execuţia comenzii se încheie;
- în prezenţa clauzei OTHERWISE, se execută grupul al n-lea de instrucţiuni, după care se trece la prima comandă de după ENDCASE.

Obs *Numai unul dintre grupurile de instrucţiuni specificate va fi executat, şi anume primul pentru care expresia logică asociată este evaluată la valoarea* adevărat.

Exemplu

```
anotimp=2
...
? 'Acest anotimp contine lunile: '
DO CASE
  CASE anotimp=1
    ?? 'Martie, Aprilie, Mai'
  CASE anotimp=2
    ?? 'Iunie, Iulie, August'
  CASE anotimp=3
    ?? 'Septembrie, Octombrie, Noiembrie'
  OTHERWISE
    ?? 'Decembrie, Ianuarie, Februarie'
ENDCASE
```

Structuri repetitive

Executarea repetată a unui grup de instrucţiuni reprezintă unul dintre principalele avantaje ale elaborării de programe, fără de care multe probleme nu ar putea fi rezolvate sau s-ar rezolva foarte greu. Structurile repetitive alcătuiesc aşa-numitele „*bucle*", care reprezintă de fapt secvenţe de comenzi executate de mai multe ori.

Buclele se împart, la rândul lor, în două grupe:

- *cu număr cunoscut de paşi* – în care un grup de instrucţiuni se execută de un număr dat de ori, cunoscut la intrarea în buclă. Comenzile folosite sunt `FOR...ENDFOR` şi `SCAN...ENDSCAN`;
- *cu număr nedefinit de paşi* – în care numărul de execuţii ale grupului de instrucţiuni este variabil, fiind dependent de o condiţie a comenzii. Pentru aceste bucle se foloseşte comanda `DO WHILE...ENDDO`.

Comanda `FOR...ENDFOR` are următoarea formă:

```
FOR <var> = <valoare start> TO <valoare finală>  STEP <increment>
  <instrucţiuni>
ENDFOR
```

Ea determină executarea repetată a grupului `<instrucţiuni>`; contorizarea acestor paşi este realizată prin variabila `<var>`. Valoarea iniţială a variabilei contor va fi `<valoare start>`. După fiecare execuţie a grupului de instrucţiuni respectiv variabila va fi incrementată sau decrementată cu o valoare constantă, `<increment>`, dacă este prezentă clauza `STEP`, sau cu 1 când `STEP` lipseşte (absenţa clauzei `STEP` este echivalentă cu `STEP 1`).

Deci, la prima execuţie a grupului de instrucţiuni, `<var>` va avea valoarea `<valoare start>`, la a doua execuţie `<valoare start>+<increment>`, la a treia execuţie `<valoare start>+<increment>+<increment>` şi aşa mai departe. Când valoarea variabilei creşte peste `<valoare finală>` (strict mai mare), în cazul unei valori pozitive a incrementului, sau scade sub `<valoare finală>` (strict mai mică) pentru o valoare negativă a incrementului, se va ieşi din buclă, programul continuând cu comanda de după `ENDFOR`.

Exemplu

Instrucţiunile:

```
FOR i=1 TO 4
  ? i
ENDFOR
```

sunt echivalente cu grupul de comenzi:

```
? 1
? 2
? 3
? 4
```

În exemplul de mai sus `i` *este variabila contor, 1 este valoarea inițială a variabilei, iar 4 este valoarea finală a acesteia. La fiecare execuție a lui* **ENDFOR** *variabila* `i` *va fi incrementată cu 1.*

La ultima execuție a comenzii `? i` *valoarea lui* `i` *va fi 4. Execuția lui* **ENDFOR** *are ca efect creșterea lui* `i` *cu 1, deci valoarea lui* `i` *va fi 5, care depășește valoarea finală, 4. Programul va continua cu prima instrucțiune de după* **ENDFOR**.

Instrucțiunile:

```
FOR i=4 TO 1 STEP -2
   ? i
ENDFOR
```

sunt echivalente cu succesiunea de comenzi

```
? 4
? 2
```

Obs *Cele trei valori,* **<valoare start>**, **<valoare finală>** *și* **<increment>**, *sunt de fapt expresii numerice ce sunt evaluate la intrarea în buclă (și nu sunt apoi reevaluate la fiecare pas). Modificarea lor în cadrul buclei nu are nici un efect asupra numărului de execuții ale instrucțiunilor buclei.*

În schimb, modificarea valorii contorului în interiorul buclei va influența numărul de execuții ale grupului de instrucțiuni, testarea variabilei contor efectuându-se la fiecare nouă execuție a grupului de instrucțiuni.

Două comenzi (clauze) speciale pot fi folosite în interiorul unei bucle FOR...ENDFOR:

- EXIT – care determină ieșirea forțată din buclă și continuarea execuției programului cu prima comandă care urmează după ENDFOR, indiferent de valoarea variabilei contor;
- LOOP – care determină saltul peste următoarele instrucțiuni ale buclei (dintre LOOP și ENDFOR), incrementarea sau decrementarea contorului și trecerea la o nouă execuție a grupului de instrucțiuni, dacă este respectată condiția de rămânere în buclă.

Exemplu

```
suma=0
FOR i=1 TO 10
  suma=suma+1
  IF i=5
    EXIT
  ENDIF
ENDFOR
```

Următorul program este identic, ca rezultat, cu cel precedent:

```
suma=0
FOR i=1 TO 10
  IF i>5
    LOOP
  ENDIF
  suma=suma+1
ENDFOR
? 'Suma este',suma,'deci s-au executat',suma,'pasi'
```

Primul program din acest exemplu funcţionează astfel: se execută comanda `suma=suma+1` *de cinci ori, pentru* `i` *având valorile 1, 2, 3, 4 şi 5. Când* `i` *ia valoarea 5, se îndeplineşte condiţia comenzii* `IF` *şi deci va fi executată comanda* `EXIT`*, care determină saltul la ultima instrucţiune a programului (cea care afişează rezultatul).*

Cel de-al doilea program din exemplul anterior are următoarea funcţionare: se execută comanda `suma=suma+1` *de cinci ori, pentru* `i` *egal cu 1, 2, 3, 4 şi 5, atâta timp cât nu este respectată condiţia comenzii* `IF`*. Când această condiţie devine adevărată, pentru* `i` *având valorile 6, 7, 8, 9 şi 10, se va executa comanda* `LOOP`*, care determină ignorarea comenzilor următoare din buclă, deci a comenzii* `suma=suma+1`*. Se va ieşi din buclă în mod normal, când* `i` *va fi 11.*

Un tip special de buclă, foarte asemănătoare cu `FOR...ENDFOR`, dar specializată în lucrul cu o tabelă, este reprezentată de comanda `SCAN...ENDSCAN`:

```
SCAN [NOOPTIMIZE]
  <domeniu> FOR <condiţie> WHILE <condiţie>
  <instrucţiuni>
ENDSCAN
```

Această comandă realizează parcurgerea tabelei curente şi executarea grupului de `<instrucţiuni>`, pentru fiecare înregistrare care aparţine domeniului specificat prin `<domeniu>`, `FOR` şi `WHILE`.

Exemplu

Având tabela ANGAJATI*, se afişează salariul brut al persoanelor de sex masculin.*

```
SCAN FOR sex=.T.
  ? sal_brut
ENDSCAN
```

Cel de-al doilea tip de buclă, cu număr nedefinit de paşi, este implementat prin intermediul comenzii DO WHILE...ENDDO:

```
DO WHILE <condiţie>
  <instrucţiuni>
ENDDO
```

Această comandă determină execuţia repetată a grupului de instrucţiuni, atâta timp cât valoarea expresiei logice <condiţie> este *adevărat*. Execuţia comenzii se desfăşoară astfel: se evaluează condiţia şi, dacă aceasta nu este îndeplinită, execuţia comenzii se încheie. Dacă este îndeplinită condiţia se vor executa instrucţiunile dintre DO WHILE şi ENDDO. La întâlnirea lui ENDDO se sare la linia cu DO WHILE şi este reevaluată condiţia. Acest ciclu continuă până când evaluarea expresiei logice conduce la valoarea *fals*, moment în care execuţia comenzii DO WHILE...ENDDO se încheie, iar programul continuă cu prima instrucţiune de după ENDDO.

Şi în această comandă pot fi incluse comenzile LOOP şi EXIT, care au aceeaşi semnificaţie ca şi la comanda FOR...ENDFOR: prima determină ignorarea restului de comenzi şi reevaluarea condiţiei, iar cea de-a doua forţează ieşirea din buclă, indiferent de valoarea expresiei logice respective.

Exemplu

Următorul program calculează suma a p numere naturale, p fiind numărul maxim pentru care suma nu depăşeşte 100.

```
suma = 0
p=1
DO WHILE suma < 100
  suma=suma+p
  p=p+1
ENDDO
suma=suma-p
a=p-1
? 'S-au adunat primele',a,'numere naturale',
? '     si s-a obtinut suma de ',suma
```

Proceduri şi funcţii definite de utilizator

Definirea şi apelarea modulelor

Execuţia unui program din interiorul altui program prin comanda DO reprezintă un pas important în structurarea aplicaţiilor de dimensiuni mai mari. Pe măsură ce creşte dimensiunea unei aplicaţii (creşte numărul liniilor de program), testarea şi depanarea acesteia devine din ce în ce mai dificilă, datorită numărului mare de variabile folosite, de instrucţiuni şi legături dintre ele. Din aceste motive s-a trecut la gruparea unor instrucţiuni în module separate, care rezolvă o anumită parte a problemei. Aceste module sunt independente între ele, comunicarea lor cu celelalte module făcându-se prin intermediul unor parametri, variabile de comunicare, care realizează interfaţa modulului cu exteriorul.

Module de acest tip pot fi create şi în interiorul fişierului ce conţine programul, formând aşa-numitele „*proceduri şi funcţii definite de utilizator*" (în engleză se foloseşte termenul **UDF** – „**U**ser **D**efined **F**unctions"). Datorită independenţei unui modul faţă de restul programului şi faţă de celelalte module, el poate fi executat de mai multe ori în cadrul unui program, prin câte o instrucţiune de apelare a modulului plasată în poziţia dorită. Se introduce astfel o modalitate de execuţie a unui grup de instrucţiuni în mai multe zone ale unui program, fără a rescrie aceste instrucţiuni la fiecare folosire.

Exemplu

Să luăm de exemplu calculul combinărilor, care se realizează pe baza formulei:

$$C_n^k = \frac{n!}{k!\,(n-k)!}$$

*Calculul factorialului n!=1*2*...*n se face cu următoarea secvenţă de instrucţiuni:*

```
n=6
FACT=1
FOR i=1 TO n
  FACT=FACT*i
ENDFOR
```

Pentru calculul combinărilor, această secvenţă se va repeta de trei ori, unde pe poziţia lui `n` *vom avea pe rând* `n`, `k` *şi* `n-k`. *Programul pentru calculul combinărilor va fi:*

```
n=6
k=3
FACT1=1
FOR i=1 TO n
  FACT1=FACT1*i
ENDFOR
FACT2=1
FOR i=1 TO k
  FACT2=FACT2*i
ENDFOR
FACT3=1
FOR i=1 TO n-k
  FACT3=FACT3*i
ENDFOR
COMBIN=FACT1/FACT2/FACT3
```

Acest mod de scriere a unui program este ineficient, pentru că acelaşi grup de instrucţiuni apare de trei ori. În situaţii ca aceasta, pentru scrierea programului se folosesc funcţii şi proceduri definite de utilizator. Folosind pentru calculul factorialului o funcţie, programul de calcul al combinărilor va căpăta următoarea formă:

```
n=6
k=3
COMBIN=FACT(n)/FACT(k)/FACT(n-k)

FUNCTION FACT
  PARAMETERS n
  FACT=1
  FOR i=1 to n
      FACT=FACT*i
  ENDFOR
  RETURN FACT
```

Primele trei linii reprezintă programul propriu-zis de calcul al combinărilor. În el se face de trei ori apel la funcţia `FACT()`, care este definită în următoarele şapte linii de program (nu ne interesează în acest moment semnificaţia instrucţiunilor din program).

Modul de definire şi de apelare a funcţiilor şi procedurilor definite de utilizator va fi tratat în cele ce urmează. O funcţie (definită de utilizator) reprezintă un grup de instrucţiuni independent, care primeşte un set de parametri de la programul apelant şi îi returnează acestuia o valoare ca rezultat al prelucrărilor efectuate asupra parametrilor transmişi. O funcţie definită de utilizator poate intra în componenţa unei expresii ca operand, ca şi funcţiile standard din Visual FoxPro.

O procedură (definită de utilizator) reprezintă de asemenea un grup de instrucţiuni ce primeşte de la programul apelant un grup de parametri, realizează

anumite prelucrări, după care redă controlul programului apelant. În schimb, o procedură nu poate intra în alcătuirea unei expresii ca operand, fiind similară comenzilor standard din Visual FoxPro.

Trebuie să se facă distincţie între definiţia funcţiei sau a procedurii şi apelul acestora. La definirea unei funcţii sau a unei proceduri se stabilesc prelucrările ce au loc în interiorul ei, parametrii care se primesc spre prelucrare şi rezultatele ce se vor transmite după prelucrare programului apelant. La apelul unei funcţii sau al unei proceduri apare doar numele care identifică respectiva funcţie sau procedură, însoţit eventual de lista parametrilor ce se transmit.

Definirea unei funcţii se face prin intermediul instrucţiunii `FUNCTION`:

```
FUNCTION <nume funcţie>
  <instrucţiuni>
ENDFUNC
```

`<nume funcţie>` reprezintă numele care se atribuie funcţiei nou definite şi care va fi folosit la fiecare apel al acesteia, pentru identificarea sa printre celelalte funcţii. Definirea procedurilor se realizează în mod analog, comanda folosită fiind `PROCEDURE`:

```
PROCEDURE <nume procedură>
  <instrucţiuni>
ENDPROC
```

Numele unei funcţii sau al unei proceduri definite de utilizator, pe care le vom numi în continuare „*rutine*" sau „*module*", poate fi alcătuit din maximum 10 caractere, începând cu literă sau cu caracterul liniuţă de subliniere şi conţinând litere, cifre sau caracterul liniuţă de subliniere.

Apelul unei funcţii se face prin numele acesteia, urmat, între paranteze rotunde, de lista parametrilor prin care se realizează comunicarea cu funcţia. La execuţie, în locul acestei construcţii se va introduce valoarea returnată de funcţie ca rezultat al prelucrărilor din interiorul său.

O procedură se execută prin comanda `DO` urmată de numele său, iar parametrii pe care programul îi transmite acesteia se introduc în lista clauzei `WITH` a comenzii:

```
DO <procedură> WITH <listă parametrii>
```

Exemplu

```
CLEAR
DO cadru
? mesaj()
```

```
FUNCTION mesaj
  RETURN 'Lucram in Visual FoxPro'
ENDFUNC

PROCEDURE cadru
  @ 10,10,14,70 BOX
ENDPROC
```

În acest exemplu, `DO cadru` determină execuţia procedurii `cadru`, definită după `PROCEDURE cadru`, iar `mesaj()` este construcţia de apelare a funcţiei `mesaj`, definită prin linia care urmează comenzii `FUNCTION mesaj`.

Procedurile şi funcţiile unui program se introduc, de regulă, după ultima instrucţiune a programului, în acelaşi fişier cu acesta.

Variabile locale şi variabile globale

Variabilele folosite într-un program sunt definite prin diferite comenzi, precum cea de atribuire, şi ele există în memorie atâta timp cât programul este în curs de rulare (la terminarea acestuia fiind eliminate automat). Probleme speciale apar atunci când un modul (program, procedură sau funcţie) este apelat din interiorul altui program. Variabilele din programul apelat sunt cunoscute în cel apelant şi invers? Ce se întâmplă când în modulul apelat se foloseşte o variabilă cu un nume identic cu al unei variabile din modulul apelant? Răspunsurile la aceste întrebări vor fi date în cele ce urmează.

La două tipuri de variabile se poate face referire într-un modul: *variabile globale* (publice) şi *variabile locale* (private). Variabilele globale sunt accesibile şi pot fi modificate în orice modul în curs de execuţie de pe un nivel inferior, egal sau superior nivelului modulului curent. Spre deosebire de acestea, variabilele locale sunt accesibile numai în modulul curent şi în cele subordonate acestuia, deci în modulele de pe niveluri mai înalte sau egale cu al modulului curent.

Pentru ca într-un modul să se declare un set de variabile private, acestea se includ în comanda `PRIVATE`:

```
PRIVATE <listă variabile>
```

Prin această comandă se declară ca fiind private variabilele din listă, dar ele nu se creează.

Variabilele globale, publice, se definesc prin comanda `PUBLIC`:

```
PUBLIC <listă variabile>
```

Spre deosebire de PRIVATE, care nu creează variabilele la care se referă, comanda PUBLIC, o dată cu declararea variabilelor respective, le şi creează. În listă se pot include şi tablouri, care sunt de asemenea create automat.

Exemplu

```
SET TALK OFF
CLEAR
PRIVATE a
PUBLIC b
a=1
b=2
DO test
NOTE aici se cunosc variabilele a,b,d
NOTE  dar nu se cunoaşte variabila c
? 'a=',a
? 'b=',b
? 'd=',d

PROCEDURE test
  PRIVATE c
  PUBLIC d
  c=3
  d=4
  NOTE aici se cunosc toate variabilele: a,b,c,d
   ? 'a=',a
   ? 'b=',b
   ? 'c=',c
   ? 'd=',d
```

Transferul parametrilor la şi de la module de program

Pentru dezvoltarea aplicaţiilor complexe, care includ mai multe programe, proceduri, funcţii etc., se recomandă ca în cadrul modulelor să se folosească numai (sau aproape numai) variabile locale, iar comunicarea cu restul modulelor să se facă prin intermediul unor variabile speciale, numite parametri.

În FoxPro sunt implementate două metode de transmitere a parametrilor la rutine:

- *prin referinţă* – în care variabila transmisă este afectată de eventualele modificări aduse în subprogram;
- *prin valoare* – când o eventuală modificare a variabilei în subprogram nu afectează valoarea acesteia în programul apelant.

Transmiterea parametrilor la un subprogram se desfăşoară în modul următor:

- se stabilesc variabilele care se vor transmite subprogramului ca parametri, într-o ordine precizată de programator la conceperea programului;
- se stabileşte un set de variabile locale subprogramului, care vor prelua valorile variabilelor transmise de la programul apelant;
- corespondenţa între variabilele transmise din programul apelant şi cele locale ale subprogramului se face prin *poziţia identică în două liste, şi anume lista cu* parametrii de apel al subprogramului şi lista variabilelor locale, specificată prin comanda `PARAMETERS`;
- în modulul apelat se lucrează cu variabilele locale respective;
- dacă tipul transmiterii este prin referinţă, la sfârşitul executării subprogramului, conţinutul variabilelor este trecut în variabilele corespunzătoare transmise ca parametri;
- dacă avem o transmitere prin valoare, ultima copiere nu mai are loc, deci în acest caz variabilele de apelare nu vor mai fi actualizate cu noile valori ale variabilelor locale corespunzătoare.

Aceasta este o reprezentare intuitivă a tehnicii de transmisie a parametrilor, în realitate folosindu-se însă metode mai complexe.

Lista variabilelor transmise ca parametri este stabilită fie prin clauza `WITH` a comenzii `DO`, în cazul apelului unui subprogram sau al unei proceduri, fie prin lista dintre parantezele rotunde ce urmează după nume, pentru un apel de funcţie. Pentru a stabili în ce variabile locale se încarcă parametrii transmişi, se foloseşte comanda `PARAMETERS`:

```
PARAMETERS <listă variabile locale>
```

Această comandă, care trebuie să fie prima comandă a unui modul (în cazul în care ea există), defineşte lista de variabile locale care vor prelua parametrii transmişi de la programul apelant. Lista variabilelor locale trebuie să aibă întotdeauna mai multe elemente (sau la fel de multe) faţă de lista parametrilor transmişi, pentru ca fiecare parametru să aibă un corespondent în subprogram.

Exemplu

```
SET TALK OFF
CLEAR
a=14
b=37
? a,'+',b,'=',suma(a,b)
c=12
d=17
prod=0
```

```
DO produs WITH c,d,prod
? c,'*',d,'=',prod

FUNCTION suma
  PARAMETERS a1,a2
  RETURN a1+a2

PROCEDURE produs
  PARAMETERS a1,b1,c1
  c1=a1*b1
  RETURN
```

Numărul de parametri transmişi programului, procedurii sau funcţiei curente este returnat de funcţia `PARAMETERS()`.

În ceea ce priveşte metoda folosită la transmiterea parametrilor, se aplică următoarele reguli:

- la programe, parametrii se transmit în mod implicit prin referinţă;
- la proceduri şi funcţii, se foloseşte implicit metoda transmiterii parametrilor prin valoare.

Pentru a forţa transmiterea unui parametru prin valoare, acesta se include între paranteze rotunde la apelul unei funcţii, iar pentru forţarea transmiterii unui parametru prin referinţă, acesta va fi precedat de caracterul '@'.

Exemplu

```
SET TALK OFF
CLEAR
STORE 1 TO a,b,c,d,e,f
alfa=test((a),@b,(c),(d),@e,@f)
? a,b,c,d,e,f

FUNCTION test
  PARAMETERS a1,a2,a3,a4,a5,a6
  STORE 2 TO a1,a2,a3,a4,a5,a6
  RETURN .T.
```

Programarea orientată spre obiecte

Capitolul 7

- De ce programare orientată spre obiecte?
- Obiecte şi clase
- Un exemplu de clasă simplă – lista
- Încapsularea
- Moştenirea
- Accesul la membrii unei clase
- Folosirea obiectelor ca membri ai unei clase
- Folosirea claselor predefinite

De ce programare orientată spre obiecte?

Trăim într-o lume dinamică, ce suferă în continuu transformări. Elementele naturii nu pot fi descrise complet printr-o serie de caracteristici, ci este necesară şi o descriere a comportării lor în diferite situaţii. Să luăm, de exemplu, o maşină, pe care o putem descrie sumar prin culoare, număr de locuri, capacitate cilindrică etc. Dar la ce bun, dacă nu ştim să o pornim şi să o conducem? În accepţiunea programării orientate spre obiecte (POO), maşina ar putea reprezenta un obiect, care ar avea în componenţă o serie de caracteristici, dar şi o serie de proceduri prin care este descrisă funcţionarea ei, modul său de utilizare etc.

Un alt exemplu de obiect, deseori întâlnit în lumea calculatoarelor, este fereastra. În accepţiunea clasică, o fereastră reprezintă o zonă de pe ecran cu anumite proprietăţi (dimensiune, culoare, conţinut etc.). Prin comenzi ale limbajului, această fereastră poate fi manipulată, deschisă, afişată, închisă etc. În POO, o fereastră reprezintă un obiect care are asociate anumite proprietăţi, dar şi anumite proceduri. Acestea din urmă sunt executate fie la comanda explicită a utilizatorului, fie la apariţia unor anumite evenimente, cum ar fi, de exemplu, un clic cu mouse-ul în interiorul ferestrei.

Unul dintre principalele elemente noi ale sistemului Visual FoxPro (faţă de versiunile anterioare de FoxPro) este introducerea tehnologiei POO, alături de varianta clasică de programare (pentru a păstra compatibilitatea cu versiunile anterioare).

Programarea orientată spre obiecte reprezintă o tehnologie modernă în domeniul programării calculatoarelor, rezultată din necesitatea realizării unor aplicaţii din ce în ce mai complexe. Programarea clasică, structurată, suferea din punctul de vedere al controlului programelor de dimensiuni mari, al reutilizării codului, al adaptării şi al extinderii unor module. Programarea structurată porneşte de la celebra ecuaţie a lui Niklaus Wirth:

Structuri de date + Algoritmi = Program

Programarea orientată spre obiecte are la bază conceptul de obiect, care reprezintă un ansamblu de date împreună cu procedurile de prelucrare a acestor date. Ecuaţia care fundamentează tehnologia POO este:

Date + Proceduri = Obiect

Prin programarea orientată spre obiecte nu se pot obţine lucruri care nu s-ar putea realiza şi prin varianta clasică de programare. Diferenţa între cele două metode constă în modul de abordare. În timp ce programarea clasică se concentrează pe prelucrări, cea orientată spre obiecte are ca punct central definirea obiectelor.

De exemplu, definirea şi activarea unei ferestre se făcea în FoxPro 2.6 (şi se poate face şi în Visual FoxPro) printr-o secvenţă de tipul:

```
DEFINE WINDOW fereastra1;
  FROM 0,0 TO 20,50;
  CLOSE FLOAT GROW MINIMIZE ZOOM SYSTEM;
  TITLE "Fereastra definita in modul clasic"
ACTIVATE WINDOW fereastra1
```

Pentru aceeaşi sarcină, în Visual FoxPro se poate folosi (şi chiar se recomandă) varianta POO, ca în secvenţa de mai jos:

```
fereastra1=CREATEOBJECT("Form")
fereastra1.caption="Fereastra definita in modul POO"
fereastra1.backcolor=16777215
fereastra1.show
```

Chiar dacă nu sunt atât de evidente, avantajele POO sunt semnificative. Printre acestea enumerăm: reutilizarea mai bună a codului, dezvoltarea mai uşoară a aplicaţiilor, controlul mai bun al programelor de dimensiuni mari etc. Deşi pentru cei familiarizaţi cu stilul clasic de programare este incomod a trece la POO fără a avea un argument solid în favoarea acestui pas, recomandăm totuşi adoptarea treptată a acestui mod de programare, deoarece el va domina viitorul (dacă nu o face deja cu prezentul).

Obiecte şi clase

La baza POO stau conceptul de obiect şi cel de clasă.

Def

*Un **obiect** reprezintă un ansamblu de date, împreună cu procedurile de prelucrare a acestora. În acest context, procedurile poartă denumirea de **metode**, iar datele obiectului se numesc **proprietăţi**.*

Definirea unui obiect se face printr-o „clasă".

Def

***Clasa** reprezintă definiţia, într-un anumit limbaj de programare, a unui tip de obiecte, adică descrierea proprietăţilor şi a metodelor componente.*

O clasă reprezintă o noţiune abstractă, atâta timp cât pe baza ei nu se creează un obiect. Crearea unui obiect presupune specificarea clasei care stă la baza sa, în acest mod identificându-se proprietăţile obiectului şi felul în care acestea sunt modificate, prelucrate.

Exemplu

Să luăm de exemplu clasa ferestrelor, care ar putea arăta schematic astfel:

```
clasă fereastră {
  proprietăţi:
       poziţie
       dimensiune
       culoare
       stare de afişare (afişată sau invizibilă)
       ...
  metode:
       metoda de afişare
       metoda de redimensionare
       metoda de schimbare a stării de afişare
       ...
}
```

Dacă avem o clasă precum cea de mai sus, nu înseamnă că avem o fereastră, ci doar că ştim prin ce se caracterizează un asemenea element. Pentru a avea efectiv o fereastră, trebuie creat un obiect pe baza clasei respective, printr-o instrucţiune de tipul:

```
obiect fereastră = Creează un obiect pe baza clasei fereastră
```

Desigur că, în exemplele de mai sus, s-au folosit pseudo-instrucţiuni, pentru a se înţelege mai bine cele două concepte. Pentru alcătuirea instrucţiunilor Visual FoxPro corespunzătoare, este necesar a fi respectate o serie de reguli de sintaxă ale limbajului.

Exemplu

De exemplu, crearea unui obiect de tip fereastră s-ar realiza printr-o instrucţiune de tipul:

```
fer = CreateObject ("Form")
```

Visual FoxPro are predefinite o serie de clase, care pot fi folosite la crearea obiectelor standard, cum ar fi ferestre, obiecte de control ale interfeţei cu utilizatorul (butoane, liste etc.), meniuri şi altele. Astfel, pentru crearea unui element de acest tip, nu mai este necesară definirea claselor respective, ci doar folosirea lor. Proprietăţile obiectelor astfel create pot fi însă modificate, ceea ce vă permite să realizaţi propriile dumneavoastră obiecte.

Pe de altă parte, limbajul Visual FoxPro permite definirea propriilor dumneavoastră clase, care să descrie obiectele folosite în programele proprii.

Un exemplu de clasă simplă – lista

Vom începe prezentarea modului de construire a claselor în limbajul Visual FoxPro printr-un exemplu simplu, o listă de numere. După cum am spus mai devreme, o clasă conţine o mulţime de date şi procedurile (metodele) prin care aceste date sunt prelucrate. Lista noastră va conţine iniţial:

- un vector `a` de, să zicem, 10 elemente, în care vom depozita elementele componente (numere);
- o variabilă numită `lungime`, în care vom memora numărul curent de valori încărcate în listă (maximum 10, datorită dimensiunii vectorului).

Ca metode ale obiectelor de tip listă vom avea:

- `adaug` – metodă pentru adăugarea de noi elemente;
- `sterg` – metodă folosită la eliminarea din listă a ultimului element introdus;
- `afisare` – metodă prin intermediul căreia sunt afişate pe rânduri succesive elementele listei.

Construirea unei astfel de liste se desfăşoară în două etape:

- mai întâi se defineşte clasa pe baza căreia se vor construi viitoarele obiecte de tip listă. Instrucţiunea folosită este `DEFINE CLASS`;
- apoi se construieşte un obiect pe baza clasei definite anterior, folosindu-se comanda `CREATEOBJECT`.

Definirea clasei se va face printr-o instrucţiune de tipul:

```
DEFINE CLASS <nume clasă> AS custom
   ...
ENDDEFINE
```

Numele noii clase definite va fi `<nume clasă>`, iar cuvântul cheie `custom` indică faptul că se defineşte o clasă a utilizatorului, care nu are o clasă părinte (nu are proprietăţi şi metode predefinite). În locul punctelor de suspensie se introduc definiţiile datelor şi metodelor membre ale clasei.

Exemplu

În exemplul nostru, definirea clasei `c_lista` se face prin următoarea secvenţă de instrucţiuni:

```
DEFINE CLASS c_lista AS custom
   DIMENSION a[10]
   lungime=0
```

```
   PROCEDURE adaug
      ...
   ENDPROC
   ...
ENDDEFINE
```

Observăm că vectorul membru al clasei `c_lista` se defineşte prin instrucţiunea `DIMENSION`, ca şi în cazul unui vector independent. La fel şi proprietatea `lungime`, membră a clasei `c_lista`, se defineşte printr-o instrucţiune de atribuire clasică.

Metodele membre ale clasei se definesc prin cuvintele cheie `PROCEDURE...` `ENDPROC`, între acestea introducându-se instrucţiunile componente ale procedurilor respective.

Exemplu

Definiţia completă a clasei este următoarea:

```
DEFINE CLASS c_lista AS custom
  DIMENSION a[10]
  lungime=0

  PROCEDURE adaug
    PARAMETERS v
    IF This.lungime<10
      This.lungime=This.lungime+1
      This.a[This.lungime]=v
    ENDIF
  ENDPROC

  PROCEDURE sterg
    IF This.lungime>0
      This.lungime=This.lungime-1
    ENDIF
  ENDPROC

  PROCEDURE afisare
    ? 'Elementele listei sunt:'
    FOR i=1 TO This.lungime
      ? 'A[',i,']=',This.a[i]
    ENDFOR
  ENDPROC

ENDDEFINE
```

În ceea ce priveşte construcţia unei proceduri membre a unui obiect, se cuvin făcute o serie de observaţii. Să luăm exemplul metodei `adaug` a clasei `c_lista`:

```
PROCEDURE adaug
  PARAMETERS v
  IF This.lungime<10
    This.lungime=This.lungime+1
    This.a[This.lungime]=v
  ENDIF
ENDPROC
```

Această procedură începe, evident, cu cuvântul cheie `PROCEDURE` şi se termină cu `ENDPROC`. Între acestea se introduce corpul procedurii, adică instrucţiunile componente. Observăm în secvenţa de mai sus că referirea la data membră `lungime` a aceluiaşi obiect căruia îi aparţine şi procedura în cauză se face prin construcţia `This.lungime`.

De fapt, sintaxa generală prin care se obţine accesul la datele şi metodele membre ale unui obiect este:

```
<obiect>.<membru>
```

Dar, cum în acest caz încă nu s-a definit obiectul `lista`, ci doar clasa `c_lista`, pentru referirea la datele şi metodele membre ale obiectului se foloseşte cuvântul cheie `This` (tradus „acesta"). Cu alte cuvinte, construcţia `This.lungime` se traduce prin *„proprietatea lungime a acestei clase"*.

Pentru a crea un obiect de tip listă, vom folosi următoarea instrucţiune:

```
lista=CreateObject("c_lista")
```

Obs

Pentru a testa acest exemplu, este necesar să se creeze un fişier text în care să se introducă instrucţiunile de mai sus. Aceasta deoarece nu se permite execuţia instrucţiunii `DEFINE CLASS` în mod interactiv, în fereastra de comenzi. În fişierul text respectiv (program), definiţia clasei se introduce la sfârşit, din cauză că blocul `DEFINE CLASS...ENDDEFINE` este tratat de Visual FoxPro la fel ca procedurile şi funcţiile, deci după acesta nu se mai pot introduce instrucţiuni executabile. Prin urmare, conţinutul programului va fi următorul:

```
lista=CreateObject("c_lista")
... (alte instrucţiuni ale programului)

DEFINE CLASS c_lista AS custom
...
ENDDEFINE
```

Desigur că, pentru cei obişnuiţi cu programarea structurată şi nefamiliarizaţi cu programarea orientată spre obiecte, programul de mai sus pare stufos, varianta clasică putând fi mai simplă. Dar, după cum am spus mai devreme, avantajele tehnologiei POO apar la aplicaţii de dimensiuni mai mari. Totuşi, chiar şi în acest exemplu, o dată construită clasa pentru listă, utilizarea ei este foarte simplă.

În acest moment, am creat obiectul `lista`, pe baza clasei `c_lista`. Să vedem acum cum putem să folosim acest obiect, cum putem să îl exploatăm. Mai întâi, trebuie să adăugăm elemente la listă, lucru care este posibil prin instrucţiuni de tipul:

```
lista.adaug(3)
lista.adaug(6)
lista.adaug(9)
...
```

Putem afişa elementele listei folosind metoda `afisare`:

```
lista.afisare
```

sau putem şterge ultimul element din listă prin comanda:

```
lista.sterg
```

Încapsularea

Def *O regulă importantă a tehnologiei POO este **încapsularea**. Conform acesteia, datele membre ale unui obiect nu pot fi modificate decât prin intermediul metodelor proprii obiectului respectiv.*

În acest fel se realizează un control strict asupra obiectului, eliminându-se eventualele surse de erori provenite din folosirea necorespunzătoare a acestor proprietăţi.

Să luăm exemplul listei din paragraful anterior. Observăm că în clasa listei au fost prevăzute procedurile de prelucrare a listelor, adică de adăugare, ştergere şi afişare. Clasa `c_lista` ar fi putut conţine doar vectorul `a` pentru memorarea elementelor componente şi variabila `lungime` pentru memorarea lungimii curente a listei. Procedurile de prelucrare ar fi putut fi construite ca proceduri independente, care să fi avut ca obiect de prelucrare datele membre ale obiectului `lista`. Dar, prin această metodă s-ar fi încălcat principiul încapsulării datelor. Cu toate acestea, sistemul nu ar fi generat o eroare, deoarece principiul este recomandat, nu impus.

Respectarea principiului încapsulării asigură o independenţă sporită a datelor faţă de programele de prelucrare. Mai precis, modificarea structurii datelor membre ale unui obiect conduce la modificarea exclusivă a metodelor membre ale obiectului respectiv (care prelucrează datele modificate), programele care utilizează obiectul rămânând nealterate.

Moştenirea

Def

***Moştenirea** reprezintă o altă facilitate specifică tehnologiei POO, conform căreia se poate construi o clasă nouă pornind de la o clasă deja existentă. Noua clasă va „moşteni" toate datele şi metodele membre ale clasei părinte, la care se vor adăuga unele noi, definite explicit în noua clasă.*

Să luăm de exemplu clasa `c_lista_max`, pe care o vom construi pornind de la clasa `c_lista`. Noua clasă va conţine, pe lângă datele şi metodele clasei părinte (`c_lista`), o nouă dată membră, numită `max`, în care vom memora valoarea maximă din listă, şi o nouă metodă, numită `maxim`, care va determina valoarea lui `max` pe baza valorilor curente ale elementelor listei.

Definirea unei clase pe baza alteia se face tot prin instrucţiunea `DEFINE CLASS`:

```
DEFINE CLASS <nume clasă nouă> AS <nume clasă părinte>
   ...
ENDDEFINE
```

Noua clasă se va numi `<nume clasă nouă>`, iar clasa părinte (de la care cea nouă moşteneşte datele şi metodele) va fi desemnată prin `<nume clasă părinte>`.

Exemplu

În cazul nostru, clasa `c_lista_max` va fi definită prin următoarea secvenţă de instrucţiuni:

```
DEFINE CLASS c_lista_max AS c_lista
  max=0

  PROCEDURE maxim
    IF This.lungime>0
      This.max=This.a[1]
      FOR i=2 TO This.lungime
        IF This.max<This.a[i]
          This.max=This.a[i]
        ENDIF
      ENDFOR
    ELSE
      This.max=0
    ENDIF
    ? 'Valoarea maxima este:',This.max
  ENDPROC

ENDDEFINE
```

Crearea unui obiect de tip `c_lista_max` se face printr-o instrucţiune de forma:

```
lista=CreateObject("c_lista_max")
```

Pentru a vedea dacă programul de mai sus funcţionează, mai întâi vom adăuga câteva elemente la listă:

```
lista.adaug(23)
lista.adaug(45)
lista.adaug(8)
...
```

Apoi vom calcula valoarea maximă prin următoarea instrucţiune:

```
lista.maxim
```

Pe ecran va fi afişată valoarea 45, care este cea mai mare dintre valorile memorate în listă.

Prin moştenire se creează, practic, ierarhii de clase, începând cu cele mai generale şi ajungându-se la cele particulare, adaptate anumitor necesităţi, rezolvând o anumită sarcină de prelucrare.

Accesul la membrii unei clase

Uneori, prelucrarea datelor unui obiect (prin metodele acestuia) poate necesita variabile temporare suplimentare, folosite în diferite operaţii interne. Aceste variabile trebuie definite tot ca proprietăţi ale obiectului (pentru a păstra independenţa obiectului faţă de exterior), dar nu trebuie puse la dispoziţia utilizatorului clasei, deoarece modificarea lor ar putea afecta prelucrările.

La fel se întâmplă şi cu metodele unui obiect. Pot exista metode care să fie folosite intern de celelalte metode ale obiectului, dar care să nu fie puse la dispoziţia utilizatorului.

Prin urmare, este necesar un mecanism de control al accesului la membrii unei clase, mecanism care să asigure protecţia membrilor la folosirea clasei de către utilizator (prin moştenire).

Din acest punct de vedere, membrii unei clase pot fi:

- **publici** – accesibili din interiorul clasei respective (în metodele membre) şi din exteriorul acesteia, din domeniul în care clasa este definită;
- **protejaţi** – accesibili atât din interiorul clasei respective, cât şi din interiorul subclaselor construite pe baza acesteia (prin moştenire);
- **ascunşi** – accesibili doar din interiorul clasei respective, nu şi din exteriorul acesteia sau din clasele derivate din ea.

În mod implicit, proprietăţile unei clase sunt publice. Pentru a declara ca private o serie de proprietăţi ale clasei, trebuie ca ele să fie incluse într-o instrucţiune PROTECTED a clasei, ca în schema de mai jos:

```
DEFINE CLASS...
  PROTECTED <proprietate1>, <proprietate2>...
  <definiţii proprietăţi>
  ...
ENDDEFINE
```

La fel se declară şi membrii ascunşi, declaraţia folosită fiind HIDDEN.

Pentru metodele unei clase, caracteristica de acces trebuie precizată chiar înaintea fiecărei definiţii. De exemplu, pentru a declara că o metodă este protejată, se foloseşte construcţia:

```
PROTECTED PROCEDURE <nume procedură>
  ...
ENDPROC
```

Exemplu

De exemplu, în definiţia clasei `c_lista`*, proprietatea* `lungime` *ar putea fi declarată ca protejată (*PROTECTED*), deoarece ea nu trebuie modificată din exterior, ci numai prin intermediul metodelor clasei (principiul încapsulării). Dacă proprietatea respectivă ar fi fost declarată ascunsă (*HIDDEN*), atunci ea nu ar mai fi fost disponibilă în clasa* `c_lista_max`*, derivată din clasa* `c_lista`*.*

Folosirea obiectelor ca membri ai unei clase

Pe lângă proprietăţi şi metode, o clasă poate avea ca membri şi alte obiecte.

Exemplu

De exemplu, o clasă a facturilor ar putea avea ca membri un obiect de tip client pentru vânzător şi unul tot de tip client pentru cumpărător. Aceasta deoarece în orice factură se completează o zonă referitoare la cumpărător şi una referitoare la vânzător.

Includerea unui obiect într-o clasă se realizează cu ajutorul construcţiei:

```
ADD OBJECT PROTECTED <nume obiect> AS <nume clasă>
```

Cuvântul cheie PROTECTED este folosit numai atunci când noul obiect (proprietăţile şi metodele sale) se doreşte a fi protejat; în caz contrar, obiectul este considerat public.

Referirea la proprietățile și metodele obiectului inclus se face, evident, prin precedarea numelor acestora de numele obiectului (separate prin punct):

```
<nume obiect container>.<nume obiect inclus>.<proprietate sau metodă>
```

Folosirea claselor predefinite

Sistemul Visual FoxPro conține o serie de clase predefinite, care pot fi folosite direct de către utilizator. De exemplu, pentru crearea unei forme nu trebuie definită întreaga clasă a acesteia, ci trebuie folosită doar clasa predefinită **Form**, care conține toate proprietățile și metodele unei forme.

Crearea unui obiect pe baza unei clase predefinite se face cu ajutorul construcției:

```
<obiect>=CreateObject(<nume clasă predefinită>)
```

deci la fel ca în cazul unei clase a utilizatorului (definită în codul respectiv), exceptând faptul că nu mai este necesară definiția clasei respective.

Exemplu

De exemplu, construcția unei forme se realizează direct prin:

```
forma=CREATEOBJECT("Form")
```

În următorul tabel sunt prezentate clasele predefinite din Visual FoxPro, fără a intra însă în detaliile de utilizare a acestora (proprietățile și metodele lor).

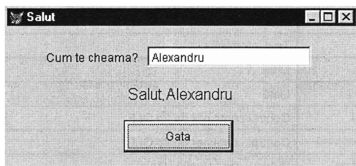
Nume clasă	Semnificație
CheckBox	Comutator
Column	Coloană într-o grilă
ComboBox	Listă derulantă
CommandButton	Buton
CommandGroup	Grup de butoane
Container	Obiect container (care poate conține alte obiecte)
Control	Obiect container, fără posibilitatea de acces la obiectele conținute

`Cursor`	Cursor (creat pe baza unei tabele sau vederi)
`Custom`	Clasă folosită pentru crearea propriilor clase ale utilizatorului
`DataEnvironment`	Mediul de date (al unei forme, al unui raport etc.)
`EditBox`	Zonă de editare a textului
`Form`	Formă
`FormSet`	Set de forme
`Grid`	Grilă
`Header`	Antet al unei grile
`Image`	Imagine
`Label`	Text informativ
`Line`	Linie
`ListBox`	Listă
`OLEControl`	Control container de tip OLE
`OLEBoundControl`	Control container de tip OLE
`OptionButton`	Butoane de selecție (radio)
`OptionGroup`	Grup de butoane radio
`Page`	Pagină a unui set de pagini alternative
`PageFrame`	Set de pagini alternative
`Relation`	Relație între două tabele (din mediul de date)
`Separator`	Separator pe o bară utilitară
`Shape`	Element semigrafic (chenar, elipsă etc.)
`Spinner`	Câmpuri de editare cu butoane de incrementare-decrementare
`TextBox`	Câmp de editare
`Timer`	Obiect de tip ceas
`ToolBar`	Bară utilitară

Exemplu

În cele ce urmează, vom prezenta un exemplu de construire a unei forme prin cod (nu cu ajutorul Constructorului de forme), pentru a ilustra modul de folosire a claselor predefinite. Menţionăm că această metodă nu este recomandată, deoarece sistemul Visual FoxPro pune la dispoziţia utilizatorilor o unealtă specială, Constructorul de forme, care este mult mai rapidă şi mai eficientă.

Forma pe care o vom construi este una simplă, folosită ca exemplu şi în capitolul referitor la Constructorul de forme (a se vedea acest exemplu).



Programul pentru construirea acestei forme este următorul:

```
* Mai intai se creeaza obiectul de tip forma
Forma1=CreateObject("Forma")

* Se afiseaza forma definita mai sus
Forma1.Show

* Se porneste procesorul de evenimente
READ EVENTS


* Aici este definita clasa formei,;
*   prin mostenire de la clasa predefinita Form

DEFINE CLASS forma AS Form
  * Proprietatile globale ale formei
  Caption="Salut"
  Height=143
  Width=359
```

```
* Aici se adauga obiectele de interfata ale formei

*   se adauga primul text informativ
ADD OBJECT Text1 AS Label
*   se adauga campul de editare
ADD OBJECT Camp1 AS TextBox
*   se adauga al doilea text informativ
ADD OBJECT Text2 AS Label
*   se adauga butonul
ADD OBJECT Buton1 AS CommandButton

* Aici se stabilesc proprietatile si metodele
*   obiectelor de interfata

* Mai intai pentru primul text informativ
Text1.Caption="Cum te cheama?"
Text1.Height=16
Text1.Left=40
Text1.Top=24
Text1.Width=104
Text1.Visible=.T.

* Apoi pentru campul de editare
Camp1.Height=21
Camp1.Left=144
Camp1.Top=20
Camp1.Width=169
Camp1.Visible=.T.

* Pentru cel de-al doilea text informativ
Text2.Caption="Salut"
Text2.Alignment=2
Text2.FontSize=12
Text2.Height=20
Text2.Left=56
Text2.Top=60
Text2.Width=244
Text2.Visible=.T.

* Pentru buton
Buton1.Caption="Gata"
Buton1.Height=33
Buton1.Left=120
Buton1.Top=96
Buton1.Width=113
Buton1.Visible=.T.
```

```
  PROCEDURE Buton1.Click
    ThisForm.Release
    CANCEL
  ENDPROC

  PROCEDURE Camp1.Valid
    ThisForm.Text2.Caption="Salut, "+ALLTRIM(This.Value)
  ENDPROC

ENDDEFINE
```

Depanarea programelor

Capitolul 8

- Introducere
- Modul de lucru cu depanatorul din Visual FoxPro
- Tehnici de depanare
  - ✓ Tipuri de rulări ale programului
  - ✓ Puncte de întrerupere
  - ✓ Controlul asupra valorilor variabilelor în timpul depanării

Introducere

Programele sunt făcute de om şi deci sunt supuse greşelilor. După ce un program depăşeşte un anumit grad de complexitate, nu se mai poate spune că el este corect 100%. Astfel, programe care aparent nu conţin erori pot „rezista" ani în exploatare, după care o eroare ascunsă să iasă la iveală şi să distrugă munca de câteva luni a mai multor persoane.

Testarea unui sistem informatic este o etapă foarte importantă în realizarea sistemului respectiv. Ea presupune trecerea sistemului prin cât mai multe situaţii posibile (de preferat toate situaţiile existente, dar acest lucru este imposibil, datorită numărului infinit de situaţii). La testare se determină anomaliile, defecţiunile şi neajunsurile în funcţionarea sistemului, care trebuie ulterior corectate. Testarea nu furnizează decât efectul eventualelor erori ale sistemului informatic, determinarea cauzei necesitând studii, uneori foarte îndelungate. Depanarea reprezintă tocmai acest proces.

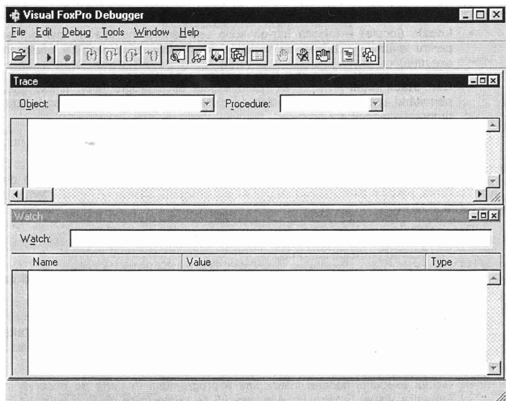
Def *Procesul de detectare şi eliminare a erorilor dintr-un program poartă numele de **depanare**.*

Orice sistem informatic folosit pentru crearea de programe şi alte sisteme informatice trebuie să ofere posibilitatea depanării programelor respective. Instrumentele pentru depanare au evoluat de-a lungul timpului, de la simple comenzi de depanare incluse în programele în lucru la adevărate depanatoare automate, care oferă soluţii şi sugestii proiectanţilor sistemelor informatice.

Depanatorul inclus în Visual FoxPro (**Debugger**) este unul performant şi, în acelaşi timp, uşor de folosit, oferind principalele facilităţi ale unui depanator, precum rularea pas cu pas a instrucţiunilor, vizualizarea dinamică a conţinutului variabilelor folosite în programul depanat etc. Depanatorul Visual FoxPro este adaptat la programele orientate spre obiecte şi la cele conduse de evenimente.

Modul de lucru cu depanatorul din Visual FoxPro

Pornirea depanatorului se realizează simplu, prin alegerea opţiunii **Debugger** (depanator) a submeniului **Tools** (unelte), care duce la deschiderea ferestrei prezentate în figura următoare:



Depanatorul este o aplicaţie independentă de Visual FoxPro. O dată pornit depanatorul, execuţia unui program în Visual FoxPro este însoţită de urmărirea sa în depanator şi invers, un program rulat în depanator îşi produce efectele în Visual FoxPro. Programul apare rulând în ambele aplicaţii (depanator şi Visual FoxPro), în depanator urmărindu-se instrucţiunile executate, iar în SGBD efectul acestora.

Depanatorul conţine mai multe ferestre, dintre care în figura de mai sus sunt ilustrate doar două, cele mai importante:

- **Trace** (urmă) – în care sunt afişate instrucţiunile executate;
- **Watch** (consultare) – în care sunt afişate valorile variabilelor folosite în programul rulat.

Alte ferestre ale depanatorului sunt:

- **Locals** (locale) – folosită într-un scop asemănător cu **Watch**, dar numai pentru anumite categorii de variabile (disponibile în procedura aflată în execuţie);
- **Call Stack** (apel stivă) – aici este afişată stiva programelor în execuţie, permiţând urmărirea programelor pe niveluri de execuţie (la apelarea unui program din altul);
- **Output** (ieşire) – în care sunt afişate ieşirile special destinate depanării, prin intermediul comenzii DEBUGOUT.

Depanatorul posedă şi o bară cu butoane, oferind căi directe spre o serie de operaţii curente, precum execuţia pas cu pas a unui program, afişarea diferitelor ferestre ale depanatorului etc.

Depanarea unui program cu ajutorul depanatorului decurge astfel:

- mai întâi se porneşte depanatorul;
- apoi în acesta se deschide pentru execuţie programul dorit (al cărui cod va apărea în fereastra **Trace** a depanatorului);
- se începe rularea prin diferite metode (pas cu pas, rulare la viteză mai mică, rularea până la un punct de oprire etc.), consultându-se în paralel în celelalte ferestre ale depanatorului diferiţi parametri ai rulării (valorile unor variabile sau expresii, evenimentele apărute, modulele în execuţie etc.);
- efectul execuţiei fiecărei instrucţiuni a programului în parte se observă în fereastra Visual FoxPro (în Windows 95 se poate trece de la o aplicaţie la alta, deci de la depanator la SGBD, cu ajutorul combinaţiei de taste Ctrl+Tab);
- prin compararea valorilor afişate cu cele dorite (aşteptate de programator) se detectează eventualele erori ale programului.

Deschiderea unui program în depanator (în vederea rulării) se realizează prin alegerea opţiunii **Open** (deschidere) a meniului **File** al depanatorului sau prin acţionarea butonului de pe bara utilitară. Din fereastra de dialog deschisă pe ecran se alege programul ce urmează a fi studiat.

În fereastra **Trace** a depanatorului vor fi afişate primele instrucţiuni ale programului, prima dintre ele având în dreptul ei o săgeată ce indică faptul că aceasta este instrucţiunea care urmează a fi executată.

Tehnici de depanare

Tipuri de rulări ale programului

În mod normal, execuția instrucțiunilor unui program este foarte rapidă, ceea ce face imposibilă urmărirea dinamică a modului de rulare a programului respectiv. De aceea, depanatorul permite alte tipuri de rulări, printre care:

- **rularea pas cu pas** – acest tip de rulare constă din execuția unei singure instrucțiuni a programului la comanda utilizatorului, după care rularea este suspendată pentru a permite proiectantului să studieze efectul instrucțiunii asupra ecranului, variabilelor etc. şi să vadă dacă este cel dorit. Execuția următoarei instrucțiuni se face la o nouă comandă a proiectantului;
- **rularea la viteză redusă** – instrucțiunile programului sunt executate continuu, numai că execuția fiecărei instrucțiuni este însoțită de o întârziere. În acest fel, programul apare ca rulând cu încetinitorul, permițând consultarea „din mers" a valorii unor variabile;
- **rularea continuă până la un anumit punct** – prin această metodă, programul este executat la viteza maximă sau încetinită până la un anumit punct, în care execuția se suspendă. Punctul respectiv poate fi stabilit manual de utilizator (prin marcarea unei anumite instrucțiuni) sau poate fi un anumit eveniment, de exemplu o expresie care are o anumită valoare sau terminarea modulului curent.

Rularea pas cu pas

Rularea pas cu pas este una dintre cele mai folosite metode de rulare pentru depanarea unui program. Execuția instrucțiunii curente se realizează prin alegerea opțiunii **Step Over** (pas peste) sau **Step Into** (pas în) a submeniului **Debug** al depanatorului. Echivalente cu **Step Over** sunt tasta F6 sau butonul {}↓ de pe bara utilitară. Acelaşi efect cu **Step Into** se obține cu ajutorul tastei F8 sau al butonului {↓} al barei utilitare a depanatorului.

Diferența dintre cele două variante este modul de tratare a modulelor definite de utilizator (programe, proceduri, funcții, metode). Dacă instrucțiunea curentă conține un apel la un modul al utilizatorului, **Step Over** va executa instrucțiunea dintr-un pas, fără intrarea în modulul respectiv, pe când **Step Into** va intra în modul şi va executa pas cu pas instrucțiunile acestuia. Vom folosi **Step Over** atunci când nu ne interesează ce se

întâmplă în modul şi **Step Into** când ceea ce se întâmplă în modul este semnificativ pentru rulare.

Obs

*De obicei, depanarea are loc astfel: la o primă rulare se realizează o trecere peste modulul în cauză (**Step Over**). Dacă rezultatele furnizate sunt corecte, atunci modulul nu mai este studiat în amănunt. Dacă însă rezultatele furnizate sunt incorecte, se realizează o nouă rulare, în care se intră în modulul respectiv (**Step Into**) pentru a se vedea unde este eroarea.*

Exemplu

Depanarea unui cod cu următoarea formă:

```
...
a=CORECT(n,i)
...

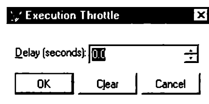
FUNCTION corect
  PARAMETERS n_1,i_1
  ...
  RETURN rezultat
```

Dacă programul de mai sus are o eroare, dar nu se ştie unde este aceasta, se procedează astfel:

- *se rulează programul pas cu pas;*
- *instrucţiunea de atribuire (`a=...`) se rulează fără intrarea în funcţia `CORECT()`, cu **Step Over**, în fereastra **Watch** urmărindu-se valoarea variabilei `a`;*
- *dacă după execuţia instrucţiunii valoarea lui `a` este cea aşteptată, înseamnă că eroarea căutată nu este în funcţia `CORECT()`;*
- *dacă însă valoarea lui `a` nu este cea aşteptată, înseamnă că eroarea este în cuprinsul funcţiei. Pentru detectarea exactă a erorii se reia rularea programului, instrucţiunea de atribuire executându-se acum cu intrarea în modul, cu **Step In**, pentru a se contina studiul în funcţia definită de utilizator.*

Rularea încetinită

Pentru rularea la viteză redusă a unui program se alege opţiunea **Throttle** a meniului **Debug** şi se specifică în fereastra deschisă pe ecran întârzierea introdusă la execuţia fiecărei instrucţiuni în parte, în secunde.



Obs *De obicei, înainte de pornirea rulării la viteză redusă a unui program, în fereastra de vizualizare a variabilelor (**Watch**) se introduc variabilele sau expresiile care urmează a fi urmărite pe parcursul rulării.*

Rularea continuă

Rularea continuă a programului deschis în depanator este realizată prin alegerea opțiunii **Resume** (reluare) a meniului **Debug**. Rularea este întreruptă de diferite evenimente (puncte de întrerupere specificate de utilizator, o anumită valoare a unei expresii etc.). Butonul corespunzător de pe bara utilitară a depanatorului pentru continuarea rulării din punctul curent este ▸|.

Rularea poate fi anulată de utilizator prin alegerea opțiunii **Cancel** (anulare) a meniului **Debug** sau prin acționarea butonului ● de pe bara utilitară.

Obs *Trebuie făcută diferența între întreruperea permanentă a execuției unui program (anularea rulării) și întreruperea temporară a execuției (adică suspendarea). În cel de-al doilea caz, există posibilitatea reluării execuției de acolo de unde a rămas, pe când în primul caz acest lucru nu este posibil.*

Puncte de întrerupere

Oprirea temporară a rulării programului se realizează cu ajutorul punctelor de întrerupere. Acestea reprezintă puncte din cadrul unui program sau evenimente cu ocazia cărora execuția programului este suspendată.

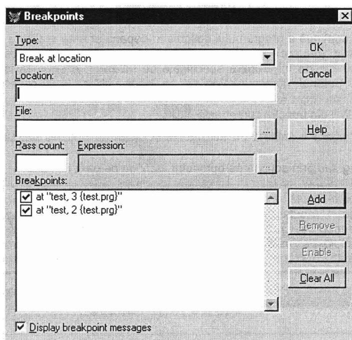
Întreruperea temporară (suspendarea) rulării unui program se poate face la apariția unuia dintre următoarele evenimente:

- ajungerea la o anumită linie a programului, marcată anterior de utilizator;

- evaluarea unei expresii la o anumită valoare;
- schimbarea valorii unei expresii;
- încheierea unui modul al programului (program, procedură, funcție, metodă etc.).

Punctele de întrerupere pot fi fie fixe, adică asociate unor linii ale programului, fie mobile (nedeterminate ca loc), dependente de îndeplinirea unei condiții și independente de poziția în care a ajuns rularea programului.

Punctele de întrerupere sunt controlate prin intermediul ferestrei de dialog **Breakpoints**, deschisă la alegerea opțiunii cu același nume a meniului **Tools** sau prin acționarea butonului de pe bara utilitară.



În lista **Breakpoints** din partea inferioară a ferestrei sunt afișate punctele de întrerupere deja stabilite (fixe sau mobile). Adăugarea unui nou punct de întrerupere în listă se face prin acționarea butonului **Add** (adăugare) și specificarea proprietăților punctului de întrerupere prin intermediul celorlalte obiecte de interfață ale ferestrei.

Ştergerea unui punct de întrerupere se face prin selectarea sa din listă, urmată de acţionarea butonului **Remove** (înlăturare).

Dezactivarea unui punct de întrerupere determină ignorarea sa de către program (punctul există, dar nu este luat în considerare). Pentru a ajunge în această stare, fie se dezactivează comutatorul din stânga punctului de întrerupere din listă, fie se selectează punctul de întrerupere respectiv, după care se acţionează butonul **Disable**. Reactivarea unui punct de întrerupere se face prin reselectarea comutatorului asociat sau prin selectarea punctului în listă şi acţionarea butonului **Enable**.

Clear All (şterge tot) este acţionat pentru ştergerea tuturor punctelor de întrerupere fixate la un moment dat. Comanda are şi un buton în bara utilitară a depanatorului, şi anume .

Rularea până la cursor

Rularea până la cursor este o tehnică prin care execuţia programului se desfăşoară în mod continuu sau încetinit şi este suspendată atunci când se ajunge la linia în care se află cursorul. Prin urmare, dacă dorim ca programul să ruleze până la o anumită linie, cel mai simplu mod de a face acest lucru este de a plasa cursorul în linia respectivă şi a da comanda corespunzătoare.

Rularea până la cursor este posibilă prin alegerea opţiunii **Run To Cursor** (execuţie până la cursor) a meniului **Debug**, prin acţionarea butonului de pe bara utilitară a depanatorului sau prin acţionarea tastei F7.

Obs *Rularea până la cursor este folosită pentru trecerea rapidă peste o porţiune a programului care nu interesează din punct de vedere al depanării şi oprirea înainte de porţiunea de interes.*

Rularea până la ieşirea din modulul curent

O altă condiţie de întrerupere a execuţiei unui program este ieşirea din modulul curent (program, procedură, funcţie sau metodă), operaţie care este realizată când se alege opţiunea **Step Out** (pas afară) a meniului **Debug**, când se acţionează butonul al barei utilitare a depanatorului sau când se acţionează combinaţia de taste Shift+F7.

Rularea până la un punct de întrerupere permanent

O linie a unui program poate fi marcată prin plasarea unui punct de întrerupere permanent, care va suspenda execuția programului atunci când se va ajunge acolo. Punctul de întrerupere este valabil până la ştergerea sa explicită (el se păstrează chiar dacă programul este închis).

Plasarea unui punct de întrerupere permanent pe o anumită linie a programului se face printr-un clic dublu în spațiul din stânga programului din fereastra **Trace**, în dreptul liniei dorite. În zona respectivă va apărea un punct roşu ce va indica prezența punctului de întrerupere. Plasarea punctului de întrerupere se poate face şi prin aducerea cursorului în linia dorită şi acționarea butonului de pe bara utilitară a depanatorului.

O dată stabilit un punct de întrerupere, el apare şi în fereastra **Breakpoints** (deschisă prin opțiunea cu acelaşi nume a meniului **Tools**) şi poate fi parametrizat în această fereastră. Lista derulantă **Type** (tip) va indica tipul *Break at location* (întrerupere la locația), câmpul de editare **Location** (locația) va indica modulul şi linia în care este plasat punctul de întrerupere, iar câmpul de editare **File** (fişier) fişierul în care este stabilit punctul de întrerupere (de fapt, fişierul în care este memorat programul).

Înlăturarea punctului de întrerupere se face printr-un nou clic dublu pe punctul roşu asociat.

Un astfel de punct de întrerupere se foloseşte atunci când se studiază un program, proces ce necesită mai multe rulări de probă. Pentru a ajunge rapid cu rularea într-un anumit punct, la o anumită situație, se plasează în linia respectivă un punct de întrerupere, rularea programului urmând a fi suspendată de fiecare dată când se ajunge la punctul respectiv.

Rularea până când este îndeplinită o anumită condiție

O altă modalitate de a condiționa întreruperea execuției unui program este prin intermediul unei expresii logice. Aceasta este evaluată la fiecare execuție a unei linii de program. Dacă evaluarea conduce la valoarea *adevărat*, execuția programului este suspendată la linia la care a ajuns în acel moment. În caz contrar, execuția continuă cu următoarea instrucțiune.

Pentru a stabili un astfel de punct de întrerupere se deschide fereastra **Breakpoints**, se adaugă noul punct de întrerupere şi apoi se alege din lista **Type** (care stabileşte tipul punctului de întrerupere) elementul *Break when expression is true* (întrerupere când expresia este adevărată). Expresia respectivă este introdusă în câmpul de editare **Expression** (expresie).

Exemplu

> *De exemplu, în cazul unei erori de tipul „împărţire la zero", am putea rula programul până când valoarea variabilei în cauză (la care se face împărţirea) se anulează.*

Rularea până când valoarea unei expresii se schimbă

O altă condiţie care ar putea conduce la suspendarea execuţiei unui program este schimbarea valorii unei variabile.

Exemplu

> *De exemplu, în cazul în care într-un program există o problemă cu o variabilă, am putea stabili un punct de întrerupere mobil, care să suspende programul ori de câte ori valoarea variabilei respective se schimbă. În acest fel, se poate studia modul în care este prelucrată variabila şi se pot descoperi schimbările nedorite ale acesteia.*

Un astfel de punct de întrerupere se stabileşte ca şi în cazul anterior (punctul de întrerupere la o anumită valoare a unei expresii), în lista **Type** a ferestrei **Breakpoints**, alegându-se însă opţiunea *Break when expression has changed* (întrerupere când expresia s-a modificat).

Rularea până când se ajunge la un punct de întrerupere şi este îndeplinită o anumită condiţie impusă de utilizator

O variantă combinată de întrerupere temporară a unui program este următoarea: rularea programului până când se ajunge la un anumit punct fix din program şi, în acelaşi timp, este îndeplinită o condiţie a proiectantului. Chiar dacă se ajunge cu execuţia în punctul respectiv, dacă nu este îndeplinită condiţia impusă, rularea programului va continua.

Caracteristică acestui tip de punct de întrerupere este opţiunea *Break at location if expression is true* (întrerupere la locaţie dacă expresia este adevărată) din lista **Type** a ferestrei **Breakpoints**.

Întreruperea unui program numai la a *n*-a trecere

O facilitate specială care se poate aplica punctelor de întrerupere fixe este întreruperea numai la a *n*-a trecere (unde *n* este o valoare dată de utilizator). Cu alte cuvinte, punctul de întrerupere respectiv va fi luat în considerare numai la a *n*-a trecere a execuției prin punctul respectiv, altfel fiind ignorat.

Acest tip de întrerupere este folosit des în studiul buclelor cu număr cunoscut de pași (FOR...ENDFOR). Dacă într-o astfel de buclă apare o eroare numai după un număr dat de pași, se pot parcurge rapid pașii corecți ai buclei și execuția se suspendă numai atunci când ne apropiem de pasul la care apare eroarea.

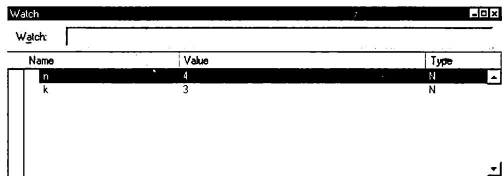
Pentru un punct de întrerupere fix, numărul de treceri permise înainte de realizarea întreruperii poate fi specificat în fereastra **Breakpoints**, în câmpul de editare **Pass count** (contor pași).

Controlul valorilor variabilelor în timpul depanării

Una dintre cele mai importante facilități oferite de orice depanator este afișarea valorii variabilelor sau a diferitelor expresii în timpul rulării. Aceste valori trebuie comparate cu cele așteptate, pentru a verifica dacă programul este corect.

Pentru aflarea imediată a valorii unei variabile, este suficientă deplasarea cursorului mouse-ului pe variabila respectivă, în programul din fereastra **Trace**. Alături de cursor apare automat un mic cadru în care se vede valoarea variabilei.

Pentru urmărirea continuă a evoluției unei variabile sau în general a unei expresii în timpul rulării unui program, se folosește fereastra **Watch** (urmărire) a depanatorului, afișată (dacă nu este deja) la alegerea opțiunii cu același nume a meniului **Window** sau la acționarea combinației de taste Alt+3.



Această fereastră este formată din două părţi:

- în partea de sus se află un câmp de editare, numit **Watch**, în care se introduc expresiile de urmărit (care se adaugă la lista din partea inferioară a ferestrei);
- în partea de jos se află o listă (sau mai precis o grilă) cu expresiile de urmărit. Coloanele listei au semnificaţiile următoare: **Name** (nume) pentru expresia urmărită, **Value** (valoare) pentru valoarea expresiei respective şi **Type** (tip) pentru tipul expresiei.

Rularea unui program în depanator este precedată de obicei de specificarea în fereastra **Watch** a variabilelor şi expresiilor de urmărit. Indiferent de tipul rulării (pas cu pas, încetinită sau continuă), în lista ferestrei **Watch** se pot observa şi compara valorile expresiilor respective cu cele aşteptate.

Câmpurile coloanelor **Name** şi **Value** pot fi modificate. Prin schimbarea conţinutului primului dintre acestea se poate schimba expresia afişată pe linia respectivă, iar prin modificarea conţinutului câmpului din coloana **Value** se poate influenţa, în timpul rulării, valoarea unei variabile.

Obs *Această din urmă tehnică prezentată este foarte puternică, dar ea trebuie utilizată cu atenţie, deoarece în acest fel este alterată execuţia normală a programului şi se poate ajunge la situaţii care nu ar fi fost posibile prin rularea normală a programului.*

Partea a IV-a – SISTEME INFORMATICE. TIPURI DE PROGRAME COMPONENTE

Programe de introducere a datelor. Constructorul de forme

Capitolul 9

- Introducere
- Constructorul de forme – mod de folosire
- Proprietăţi şi metode ale formelor în ansamblu
- Obiecte de interfaţă ce pot fi incluse în forme
  - ✓ Textele informative (**Label**)
  - ✓ Imagini, linii şi chenare
  - ✓ Câmpurile de editare (**Text Box**)
  - ✓ Zonele de editare a textului (**Edit Box**)
  - ✓ Câmpuri de editare cu butoane de incrementare-decrementare (**Spinner**)
  - ✓ Butoane şi grupuri de butoane (**Command** şi **Command Group**)
  - ✓ Butoane radio sau butoane de selecţie (**Radio Button**)

- ✓ Comutatoare (**Check Box**)
- ✓ Liste (**List Box şi Combo Box**)
- ✓ Grile (**Grid**)
- ✓ Pagini alternative (**Page Frame**)
- ✓ Ceas (**Timer**)
- ✓ Bare utilitare (**Toolbar**)

❖ Tehnici speciale folosite la construirea formelor

Introducere

Ce este o formă şi la ce se foloseşte?

Introducerea datelor reprezintă una dintre principalele operaţii ale gestiunii bazelor de date, de obicei cea mai mare consumatoare de timp. În multe cazuri, utilizatorul petrece o mulţime de timp introducând datele în bazele de date, pentru ca apoi să le extragă în doar câteva secunde. Partea sistemului informatic cu care utilizatorul vine în contact reprezintă „interfaţa".

Interfaţa cu utilizatorul a unui sistem informatic este alcătuită din mai multe tipuri de elemente, precum meniuri, ferestre, obiecte de interacţiune cu utilizatorul etc.

Def

***Forma**, sau **formularul**, reprezintă principalul element al interfeţei cu utilizatorul a sistemelor informatice, folosit în cadrul etapei de introducere a datelor.*

Din punct de vedere tehnic, formele reprezintă un ansamblu de elemente (obiecte de interfaţă, proceduri şi funcţii, proprietăţi etc.), folosite împreună pentru preluarea de la utilizator a unor parametri necesari rulării programului.

În variantele mai vechi de SGBD, termenul folosit pentru forme era „ecrane de introducere a datelor", deoarece introducerea datelor necesare rulării unui program se făcea în ecrane distincte, afişate pe rând pe ecranul monitorului. O dată cu evoluţia interfeţelor cu utilizatorul, ecranele de introducere s-au transformat în ferestre, care erau afişate pe ecranul monitorului atunci când se citeau date în ele.

Ecranele de introducere a datelor erau implementate prin intermediul „programelor de introducere a datelor". Cu alte cuvinte, pentru a construi un astfel de ecran, trebuia să se introducă într-un fişier (program) comenzile de definire a diferitelor elemente ale ecranului. Când se dorea citirea datelor, se lansa în execuţie programul respectiv, care afişa la monitor elementele ecranului de introducere a datelor.

O dată cu apariţia tehnologiei POO, ecranele de introducere a datelor au fost înlocuite cu forme. Acestea reprezintă obiecte care au asociate proprietăţi şi metode, ce stabilesc aspectul exterior şi comportamentul în diferite situaţii.

O formă apare pe ecran ca o fereastră clasică, dar ea reprezintă mai mult decât atât. Fiind un obiect (în sensul POO), forma este caracterizată prin anumite atribute, proprietăţi, ce pot fi modificate de utilizator după dorinţă. Aceste proprietăţi dau aspectul formei şi comportamentul în diferite situaţii. De exemplu, o formă poate avea ca proprietăţi poziţia sa pe ecran sau culoarea fondului.

Pe lângă proprietăţi, o formă poate avea ataşate secvenţe de cod (numite „metode" în cadrul POO) executabile în diferite ocazii. Modelul programării orientate

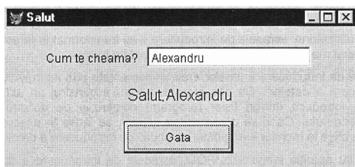
spre obiecte implementat în Visual FoxPro este şi unul condus de evenimente. Aceasta înseamnă că sistemul interceptează anumite evenimente (precum apăsarea unei taste, mişcarea mouse-ului etc.) şi transmite formei un semnal prin care o informează despre eveniment. Dacă la proiectarea formei s-a specificat o secvenţă de cod pentru evenimentul respectiv, aceasta va fi executată. Astfel, utilizatorul poate specifica modul în care forma reacţionează la evenimentele exterioare, deci poate stabili comportamentul său.

O formă nu reprezintă o simplă fereastră cu proprietăţi şi metode ataşate evenimentelor, ci ea poate conţine obiecte de interfaţă, prin intermediul cărora utilizatorul introduce datele solicitate de program. Aceste obiecte de interfaţă sunt implementate tot prin intermediul tehnologiei POO, prin urmare ele vor fi caracterizate prin proprietăţi şi metode ataşate diferitelor evenimente.

Formele sunt folosite acolo unde programul comunică cu utilizatorul, în ambele sensuri: afişarea pe ecran a unor date, pentru informarea utilizatorului cu privire la rezultatul unor prelucrări, şi citirea de la acesta a unor parametri necesari rulării programului respectiv.

Exemple de forme

Un exemplu banal de formă este următorul:



În câmpul de editare al acestei forme, utilizatorul îşi poate introduce numele, după care, în forma respectivă este afişat un mesaj de salut. La acţionarea butonului **Gata**, forma este eliminată de pe ecran. Acest exemplu va fi discutat într-un alt paragraf.

Un exemplu mai complex de formă este cea pentru introducerea datelor într-o bază de date a facturilor emise şi primite de o unitate economică. Aceasta ar putea arăta astfel:

Factura

de marfa emisa Numar 1 din 02/05/1998 Partener ALFA

Seria BDC13456801B

Nr.aviz insotire marfa 23421

Inregistrari | Termene | Plati | Contari | Expeditie

Cod	Denumire produs	U.M.	Cantitate	Pret unitar	Valoare	TVA
bere	Bere Gambrinus	buc	1240.00	2320.00	2876800.00	632896.00

Cantitatea din stoc din produsul selectat 0.00

Factura de retur   Reducere de pret 0.00 v   Total 2876800.00 632896.00

Total de plata 3509696.00

Anulata   Nom.Prod.   Nom.Part.   Salvare   Tiparire   Stergere   Terminare

Această formă este complexă, ea conţinând o mulţime de elemente, fiecare cu caracteristici speciale. De asemenea, forma răspunde la o mulţime de evenimente într-un mod specific, ceea ce creşte semnificativ complexitatea sa.

Cum se construieşte şi cum se foloseşte o formă?

Pentru a fi folosită, o formă trebuie mai întâi creată. Aceasta înseamnă că proiectantul trebuie să specifice caracteristicile formei, elementele sale componente şi proprietăţile lor. Toate aceste date sunt depozitate pe disc într-un fişier cu o structură specială.

O dată construită o formă, ea poate fi folosită pentru citirea unor date de la utilizator. Pentru aceasta, este lansat în execuţie un modul special al sistemului, care citeşte din fişierul formei toate elementele acesteia şi le afişează pe ecranul monitorului, în funcţie de caracteristicile precizate anterior, în etapa de creare.

Obs *Prin urmare, o formă se creează mai întâi, o singură dată, după care, ori de câte ori se doreşte folosirea ei, ea se „execută".*

Comanda pentru rularea unei forme este:

```
DO FORM <nume formă>
```

Ca urmare a acestei comenzi, sistemul afişează pe ecran forma respectivă şi permite utilizatorului interacţiunea cu ea (introducerea datelor în câmpurile componente ale formei).

Obs *În Visual FoxPro, caracteristicile unei forme sunt memorate într-o tabelă cu o structură specială. Fiecare element al formei este memorat într-o înregistrare a tabelei respective. Spre deosebire de programele clasice de introducere a datelor, care erau de fapt înşiruiri de comenzi pentru definirea elementelor ecranelor, formele din Visual FoxPro reprezintă fişiere de date (tabele) în care sunt memorate caracteristicile elementelor formei şi care sunt interpretate de un modul special al sistemului. Prin urmare, comanda de „rulare" a unei forme nu execută comenzile de definire a formei respective, ci lansează în execuţie modulul pentru interpretarea datelor caracteristice ale formei.*

Crearea unei forme se poate realiza în două moduri:

- fie prin includerea într-un program a comenzilor necesare definirii elementelor formei;
- fie prin construirea unei tabele în care se vor memora caracteristicile formei şi care va fi interpretată de modulul special al sistemului (menţionat anterior).

La rândul ei, metoda comenzilor incluse într-un program pentru construirea unei forme are două variante:

- prin comenzi clasice de definire a elementelor formei;
- prin comenzi POO echivalente.

Această ultimă variantă este posibilă datorită includerii în Visual FoxPro a tehnologiei POO (programarea orientată spre obiecte) şi are o serie de avantaje, subliniate în capitolul referitor la acest subiect.

Exemplu

De exemplu, definirea unei ferestre se poate face prin următoarea secvenţă de comenzi:

```
DEFINE WINDOW fer1;
  FROM 4,10 TO 16,60;
  CLOSE FLOAT MINIMIZE ZOOM SYSTEM GROW;
  TITLE "Fereastra clasica"
```

Varianta POO echivalentă este următoarea:

```
fer1 = CREATEOBJECT("Form")
fer1.Caption = "Forma definita prin POO"
fer1.Show
```

Visual FoxPro este compatibil cu sistemele FoxPro anterioare. Prin urmare, vechile metode de definire a ecranelor de introducere a datelor vor funcţiona şi în noul SGBD, dar pentru programe create în Visual FoxPro se recomandă variantele noi, mai performante.

Cea de-a două metodă de construire a unei forme este cea a construirii unei tabele în care să se memoreze caracteristicile formei. Acest lucru se poate realiza manual, dar ar necesita cunoaşterea amănunţită a structurii tabelei şi a altor date tehnice specifice sistemului.

Pentru a elimina acest inconvenient, proiectanţii sistemului Visual FoxPro au prevăzut un utilitar special cu ajutorul căruia utilizatorii pot crea rapid forme. Acest utilitar permite definirea interactivă a caracteristicilor formei, a elementelor sale componente şi a caracteristicilor acestora, toate detaliile tehnice de codificare şi memorare fiind preluate de sistem. Utilitarul se numeşte **Constructorul de forme** şi va face în continuare obiectul acestui capitol.

Pornirea Constructorului de forme pentru crearea unei noi forme se realizează cu ajutorul comenzii:

```
CREATE FORM <nume formă>
```

O dată pornit Constructorul de forme, în fereastra sa de lucru este deja definită o fereastră, care are o serie de proprietăţi şi metode implicite. Acestea însă pot fi modificate ulterior de utilizator, care poate astfel să-şi configureze după dorinţă forma respectivă. Acelaşi principiu se aplică şi obiectelor incluse într-o formă: o dată cu definirea lor, proprietăţile acestora capătă valori implicite, care pot fi ulterior modificate pentru obţinerea aspectului şi comportamentului dorit de utilizator.

Prin urmare, construirea unei forme se reduce la definirea sa şi a obiectelor de interfaţă componente şi apoi la modificarea proprietăţilor şi metodelor asociate în conformitate cu aspectul şi comportamentul dorit.

Referirea la proprietățile și metodele asociate. Ierarhia obiectelor dintr-o formă

Modificarea valorii unei proprietăți a unei forme se poate face fie printr-o metodă interactivă (a se vedea mai departe fereastra de proprietăți a Constructorului de forme), fie printr-o comandă de atribuire. Pentru această din urmă metodă, este necesară referirea la proprietatea respectivă, care se face prin construcția:

```
<nume formă>.<nume proprietate>
```

Exemplu

*De exemplu, proprietarea **Caption** (titlu) a formei **Form1** se modifică printr-o instrucțiune de atribuire de tipul:*

```
Form1.Caption = "Titlul formei"
```

La fel se apelează și metodele asociate unei forme.

Exemplu

*De exemplu, execuția metodei **afisare** a formei **Form1** se realizează cu ajutorul comenzii:*

```
Form1.Afisare
```

Formele pot conține, la rândul lor, obiecte. Referirea la un obiect din cadrul unei forme se realizează cu ajutorul unei construcții asemănătoare, adică:

```
<nume formă>.<nume obiect>
```

Dacă se dorește modificarea unei proprietăți a unui obiect inclus într-o formă, se va folosi construcția:

```
<nume formă>.<nume obiect>.<proprietate> = <valoare nouă>
```

Exemplu

*De exemplu, presupunând că forma **Form1** conține un text explicativ numit **Label1**, modificarea textului atașat acestui obiect se face prin atribuirea unei noi valori proprietății **Caption** a obiectului, lucru care se realizează printr-o comandă de tipul:*

```
Form1.Label1.Caption = "Noul text"
```

Există, de asemenea, obiecte complexe care pot conține, la rândul lor, alte obiecte. Acesta este, de exemplu, cazul unei grile, care reprezintă de fapt un tabel cu diferite obiecte, de obicei câmpuri de editare. Referirea la obiectele incluse în alte obiecte complexe se face prin precedarea numelor acestora cu numele obiectului care le conține și separarea lor printr-un punct.

Exemplu

*De exemplu, dacă grila **Grid1** conține două coloane numite **Column1** și **Column2**, iar noi dorim modificarea proprietății **Margin** a câmpului de editare **Text1** din coloana **Column2**, vom folosi construcția:*

```
Form1.Grid1.Column2.Text1.Margin = <valoare nouă>
```

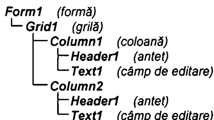
Prin includerea obiectelor unele în altele (acolo unde este cazul), se obține de fapt o ierarhie de obiecte. Referirea la un obiect dintr-o asemenea ierarhie se face prin precedarea numelui obiectului respectiv cu numele obiectelor care îl conțin, începând de la obiectul cel mai cuprinzător (forma), spre obiectele din interior.

Există câteva construcții speciale care permit scurtarea construcției folosite la desemnarea obiectelor din cadrul unei ierarhii. Acestea sunt:

- `This` – care se referă la obiectul curent;
- `ThisForm` – care indică forma curentă (ce conține obiectul curent);
- `ThisFormSet` – care desemnează setul de forme curent (ce conține forma curentă);
- `Parent` – care se referă la obiectul care conține obiectul curent (obiectul imediat superior în ierarhia de obiecte).

Exemplu

De exemplu, dacă avem următoarea ierarhie de obiecte:



*și dorim ca în cadrul metodei asociate evenimentului **Valid** al obiectului **Text1** din coloana **Column1** a grilei **Grid1** a formei **Form1** să modificăm titlul*

*(proprietatea **Caption**) obiectului **Header1** al aceleiaşi coloane, putem folosi următoarele construcţii:*

```
Form1.Grid1.Column1.Header1.Caption = "Noul Titlu"

This.Parent.Header1.Caption = "Noul Titlu"
```

Aceste cuvinte cheie ne oferă, pe lângă posibilitatea de scurtare (uneori) a construcţiei de referire la anumite obiecte, şi avantajul referirii relative la anumite obiecte, proprietăţi şi metode. De exemplu, a doua construcţie prezentată în exemplul anterior se poate folosi şi pentru obiectul **Text1** din coloana **Column2**, caz în care se va modifica titlul lui **Header1** din coloana **Column2**.

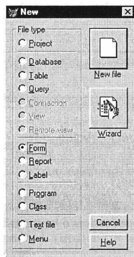
Constructorul de forme – mod de folosire

Cum se porneşte Constructorul de forme?

Am văzut deja cum poate fi pornit Constructorul de forme, prin introducerea în fereastra de comenzi a sistemului Visual FoxPro a comenzii:

```
CREATE FORM <nume formă>
```

Altă metodă este cea interactivă: se alege opţiunea **New** (nou) a meniului **File** (fişier), apoi, din fereastra de dialog afişată pe ecran, se alege butonul **Form** (formă), indicându-se astfel tipul de element ce urmează a fi creat.

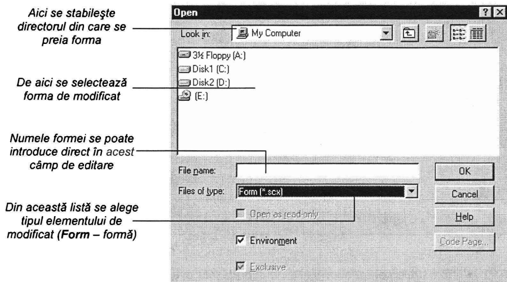


După acționarea butonului **New file** (fişier nou) urmează precizarea numelui formei într-o fereastră de dialog specifică.

Modificarea unei forme create se face cu ajutorul comenzii:

```
MODIFY FORM <nume formă>
```

sau prin alegerea opțiunii **Open** a meniului **File**. Pe ecran va fi deschisă o fereastră în care se va selecta forma ce urmează a fi modificată.

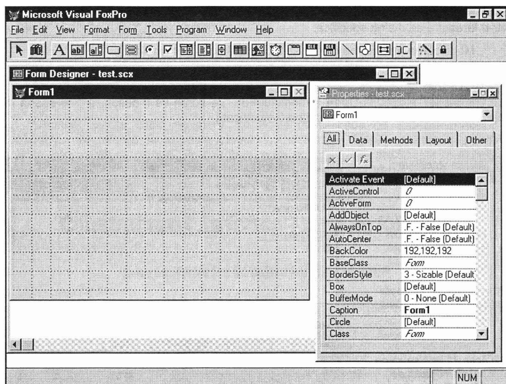


Primul ecran. Elemente componente

O dată lansat în execuție Constructorul de forme, pe ecran este afişată fereastra acestuia, care ar putea arăta ca în figura următoare.

Constructorul de forme conține următoarele elemente:

- *fereastra principală* – în care este afişată forma (în figură, ea poartă numele de **Form Designer**);
- *fereastra de proprietăți* – care permite modificarea diferitelor proprietăți ale elementelor selectate în fereastra principală a utilitarului (în figură, ea apare cu titlul **Properties**);



- *ferestrele pentru secvențele de cod ataşate evenimentelor* – care folosesc la precizarea secvențelor de cod ce urmează să fie executate la apariția diferitelor evenimente în cadrul formei (acționarea unui buton, selectarea unui element etc.). În figură nu apar;
- *fereastra mediului de date al formei* – numită **Data Environment**, ascunsă momentan în figură, folosită la precizarea tabelelor care vor fi deschise automat în momentul definirii formei şi a relațiilor între acestea;
- *o serie de bare utilitare* – folosite pentru manipularea obiectelor formei. Principalele bare utilitare sunt:
  - bara utilitară pentru obiecte de interfață (**Form Controls Toolbar**), din care se aleg obiectele ce vor fi definite în cadrul unei forme. În figură apare în partea superioară a ferestrei;
  - bara utilitară pentru manipularea obiectelor (**Layout Toolbar**) – se foloseşte pentru executarea diferitelor operații cu obiectele formei;

- bara utilitară pentru controlul culorilor (**Color Palette**).

Nu toate aceste elemente apar pe ecran la pornirea Constructorului de forme. La un moment dat pot fi afişate unele dintre ele, ulterior acestea pot fi ascunse şi afişate altele. De asemenea, poziţia pe ecran a elementelor nu este fixă; ele pot fi mutate de utilizator în funcţie de preferinţele sale.

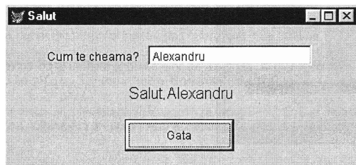
Afişarea unui anumit element pe ecran se face prin alegerea opţiunii corespunzătoare din submeniul **View** (vizualizare), astfel:

- **Properties** – pentru afişarea ferestrei de proprietăţi;
- **Code** – pentru afişarea unei ferestre pentru o secvenţă de cod;
- **Data Environment** – pentru afişarea ferestrei mediului de date;
- **Form Controls Toolbar** – pentru afişarea barei utilitare a obiectelor de interfaţă;
- **Layout Toolbar** – pentru afişarea barei utilitare de manipulare a obiectelor;
- **Color Palette Toolbar** – pentru afişarea barei utilitare de manipulare a culorilor.

Ascunderea unui anumit element se face prin acţionarea butonului de închidere din partea din dreapta-sus a ferestrei respective.

Modul de lucru cu elementele Constructorului de forme. Exemplu de formă simplă

Modul de lucru cu elementele puse în evidenţă mai sus va fi prezentat pe un exemplu concret, acela al unei forme simple pentru afişarea unui salut către utilizator.



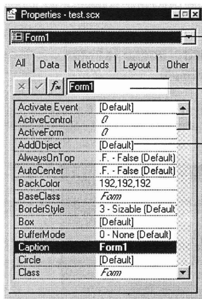
Vom începe cu pornirea Constructorului de forme, aşa cum s-a precizat anterior. În fereastra principală a utilitarului observăm că deja a fost definită fereastra formei respective. Însă dimensiunile şi caracteristicile acesteia pot fi modificate.

Vom stabili apoi dimensiunile formei. Pentru aceasta, se deplasează cursorul pe chenarul din dreapta formei şi se trage mouse-ul (cu butonul stâng apăsat) într-o nouă poziţie. Menţionăm că, la rulare, forma va apărea pe ecran exact aşa cum se vede în fereastra Constructorului de forme.

La fel se poate proceda şi cu celelalte laturi ale chenarului formei. Dacă se doresc dimensiuni exacte ale formei, se poate folosi fereastra de proprietăţi a Constructorului de forme, în care se vor modifica valorile proprietăţilor:

- **Height** pentru înălţime;
- **Width** pentru lăţime.

Dacă fereastra de proprietăţi nu este afişată pe ecran, pentru a realiza acest lucru trebuie aleasă opţiunea **Properties** a meniului **View**. O altă variantă pentru această operaţie ar putea fi alegerea opţiunii cu acelaşi nume a meniului afişat ca urmare a unui clic cu butonul drept al mouse-ului pe formă.



În partea superioară a ferestrei de proprietăţi se află o listă în care sunt afişate obiectele de interfaţă ale formei. Proprietăţile şi metodele afişate în lista principală a

ferestrei (cea de jos) se referă la obiectul care este curent selectat în lista derulantă din partea superioară a ferestrei. Prin urmare, modificarea unei proprietăţi a unui obiect de interfaţă trebuie precedată de selectarea obiectului respectiv în lista derulantă din partea superioară a ferestrei de proprietăţi. Acest lucru se poate realiza prin selectarea simplă a obiectului dorit din lista derulantă (evident, pentru aceasta fiind necesar să cunoaştem numele lui), dar se poate realiza şi printr-un clic simplu direct pe obiectul dorit din forma afişată în fereastra de lucru a Constructorului.

O dată selectat obiectul de interfaţă, pentru modificarea unei proprietăţi a acestuia se selectează proprietatea respectivă în lista din partea de jos a ferestrei de proprietăţi. Valoarea curentă a proprietăţii respective va fi afişată în câmpul de editare al ferestrei, câmp în care se va introduce noua valoare.

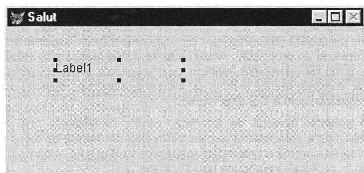
Obs *În funcţie de tipul proprietăţii modificate, câmpul de editare se transformă în listă sau în alt tip de obiect de interfaţă, pentru a facilita alegerea variantei dorite de către utilizator. De exemplu, în cazul proprietăţii **Alignment** (aliniere) există doar trei valori valide, şi anume: la stânga, la dreapta şi centrat; prin urmare, un obiect de tip listă este mai potrivit decât un câmp de editare clasic.*

După stabilirea dimensiunilor formei se trece la stabilirea titlului său, în exemplul nostru „Salut". Acest lucru se face simplu, modificându-se proprietatea **Caption** a formei (în fereastra de proprietăţi).

Urmează definirea obiectelor de interfaţă ale formei, şi anume:

- un text informativ („label" în engleză, tradus „etichetă"), în care se introduce întrebarea „Cum te cheamă?";
- un câmp de editare, în care utilizatorul îşi va introduce numele;
- un text informativ conţinând salutul către utilizator;
- un buton numit „Gata", ce va fi folosit la închiderea formei, atunci când utilizatorul doreşte terminarea lucrului cu forma respectivă.

Pentru a defini în cadrul formei un text informativ, mai întâi se acţionează butonul **A** (label) de pe bara utilitară a obiectelor de interfaţă. Dacă aceasta nu este afişată pe ecran, trebuie aleasă opţiunea **Form Controls Toolbar** din meniul **View**. Apoi se trasează cu mouse-ul poziţia şi dimensiunea noului obiect în fereastra formei (clic pe colţul din stânga-sus al obiectului şi tragere cu mouse-ul până în poziţia în care se doreşte plasarea colţului din dreapta-jos al acestuia). Forma va arăta astfel:



Poziția (relativ la formă) și dimensiunea obiectului pot fi modificate și ulterior, fie prin tragerea cu mouse-ul a colțurilor și a laturilor acestuia (a se vedea pătrățelele negre ce delimitează obiectul), fie în fereastra de proprietăți, prin modificarea parametrilor:

- **Left** – poziția laturii stângi a obiectului relativ la latura stângă a formei;
- **Top** – poziția laturii superioare a obiectului relativ la latura superioară a formei;
- **Width** – dimensiunea orizontală a formei (lățimea);
- **Height** – dimensiunea verticală a formei (înălțimea).

Urmează specificarea textului afișat în obiect (în locul lui „Label1", care este afișat în mod implicit); acest lucru se realizează în fereastra de proprietăți, în câmpul **Caption**. Textul va fi „Cum te cheamă?".

Vom defini cel de-al doilea obiect de interfață al formei, și anume câmpul de editare în care utilizatorul își va introduce numele. Această operație debutează cu acționarea butonului abl (**Text Box**) de pe bara utilitară a obiectelor de interfață, urmată de trasarea pe formă a obiectului respectiv: clic simplu în poziția colțului din stânga-sus al câmpului și apoi tragere cu mouse-ul până în poziția colțului din dreapta-jos al câmpului de editare. Momentan, pentru acest obiect nu vom modifica nici o proprietate.

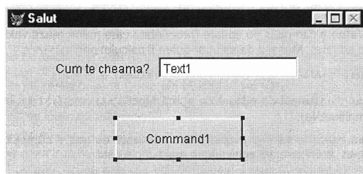
Următorul obiect care se va defini în cadrul formei este un text informativ, în cadrul căruia va fi afișat salutul către utilizator. Acest obiect va fi definit asemănător cu textul informativ anterior (primul obiect definit), însă pentru el se va modifica proprietatea **Caption** (textul afișat) la o valoare nulă (șirul vid), deoarece inițial, aici nu va apărea nimic (numai după introducerea numelui va fi afișat salutul respectiv).

O altă caracteristică a celui de-al doilea text informativ al formei Salut este fontul diferit față de primul text. În primul caz s-a păstrat fontul implicit în care este afișat textul,

şi anume Arial cu dimensiunea de 9. Afişarea salutului se va face cu un font mai mare, pentru aceasta modificându-se proprietatea **FontSize** (dimensiune font) de la valoarea 9 la 12.

De asemenea, pentru acest text informativ trebuie schimbată alinierea, din valoarea „la stânga" în „centrată". Aceasta înseamnă că proprietatea **Alignment** (aliniere) va lua valoarea 2 – *Center* (valoarea implicită este *1 – Left*, adică aliniere la stânga).

Ultimul obiect definit în forma Salut este butonul Gata, cu ajutorul căruia se realizează închiderea formei. Acest buton se defineşte apăsând butonul (**Command Button**) şi stabilind pe formă, cu mouse-ul, poziţia şi dimensiunea noului buton. Forma va arăta astfel:



Titlul noului buton va fi Gata, modificarea realizându-se prin atribuirea valorii „Gata" proprietăţii **Caption** a butonului.

Dacă până acum am stabilit aspectul exterior al formei Salut, este momentul să trecem la configurarea comportamentului acesteia, adică la stabilirea modului în care forma va răspunde la diferite evenimente externe.

În această formă trebuie interceptate două evenimente:

- terminarea introducerii numelui utilizatorului în câmpul de editare al formei, moment în care trebuie afişat mesajul de salut;
- terminarea lucrului cu forma, eveniment declanşat de acţionarea de către utilizator a butonului Gata.

Fiecare obiect al unei forme (şi forma însăşi) poate răspunde la o serie de evenimente predefinite, serie care însă nu poate fi modificată (nu pot fi ataşate alte evenimente şi nici nu pot fi şterse). În schimb, se poate modifica răspunsul obiectelor la

evenimentele respective, prin ataşarea unei secvenţe de cod pentru fiecare eveniment la care obiectul răspunde diferit faţă de răspunsul implicit, stabilit de sistem.

Primul dintre evenimentele amintite mai sus este evenimentul **Valid** (validare) al câmpului de editare al formei (evenimentul apare atunci când se termină introducerea unei valori într-un câmp de editare). Pentru ca la producerea acestui eveniment să apară mesajul de salut, vom introduce în secvenţa de cod asociată evenimentului o comandă care să modifice textul afişat în cadrul celui de-al treilea obiect al formei (numit **Label2**), textul informativ al salutului.

Prin urmare, acestui eveniment îi vom ataşa comanda de modificare a proprietăţii **Caption** a textului informativ **Label2**. Comanda respectivă este:

```
ThisForm.Label2.Caption = <mesajul de salut>
```

Să vedem cum alcătuim mesajul de salut. Acesta este format din şirul de caractere „`Salut, `" şi din numele abia introdus de utilizator. Acest nume se găseşte în proprietatea **Value** a câmpului de editare (proprietate care memorează valoarea curentă din câmpul de editare). Mesajul de salut ar putea fi calculat prin expresia:

```
"Salut, "+ALLTRIM(ThisForm.Text1.Value)
```

(asupra textului din câmpul de editare se aplică funcţia `ALLTRIM()`, care elimină din şir spaţiile nesemnificative).

Dar cum expresia va fi introdusă într-o secvenţă de cod a câmpului de editare, `ThisForm.Text1`, o construcţie echivalentă mai simplă este:

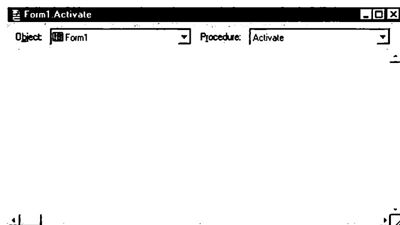
```
"Salut, "+ ALLTRIM(This.Value)
```

Prin urmare, comanda pentru afişarea mesajului de salut este:

```
ThisForm.Label2.Caption = "Salut, "+ALLTRIM(This.Value)
```

Ataşarea acestei comenzi la evenimentul **Valid** al câmpului de editare se face astfel:

- mai întâi se selectează obiectul respectiv;
- apoi se deschide o fereastră a Constructorului de forme pentru secvenţe de cod (a se vedea figura următoare), operaţie realizată prin alegerea opţiunii **Code** a meniului **View**;
- din lista **Object** din partea superioară a ferestrei se alege obiectul **Text1**, indicându-se astfel că se doreşte modificarea unei metode a acestui obiect;
- din lista **Procedure** se alege elementul **Valid**, indicându-se în acest mod că urmează a fi modificată secvenţa de cod ataşată evenimentului **Valid**;
- în zona de editare a acestei ferestre se introduce comanda respectivă.



Alt mod de afişare a unei ferestre de cod a Constructorului de forme este prin execuţia unui clic cu butonul drept al mouse-ului pe obiectul a cărui metodă se doreşte a fi modificată şi alegerea opţiunii **Code** din meniul afişat pe ecran. Ca urmare a acestei operaţii, este afişată fereastra de cod, dar, în acelaşi timp, este selectat automat în lista **Object** obiectul de interfaţă respectiv.

Un al doilea comportament ce trebuie configurat în forma Salut este închiderea formei la acţionarea butonului Gata. Evenimentul respectiv este **Click**, asociat butonului respectiv (ori de câte ori se execută un clic simplu pe un obiect de interfaţă, sistemul generează un eveniment **Click** pentru obiectul respectiv).

Pentru acest eveniment asociem comanda de închidere a formei, care este:

```
ThisForm.Release
```

(**Release** este metoda prin care forma se închide).

Cu aceasta, etapa de construire a formei s-a încheiat. Forma poate fi salvată pe disc prin alegerea opţiunii **Save** (salvare) a meniului **File**. Oricum, încercarea de închidere a Constructorului de forme neprecedată de salvare conduce la afişarea unei ferestre de avertizare, din care se poate declanşa direct salvarea.

Rularea formei este realizată prin introducerea în fereastra de comenzi a sistemului a comenzii:

```
DO FORM salut
```

Proprietăţi şi metode ale formelor în ansamblu

În acest paragraf ne propunem să evidenţiem câteva dintre proprietăţile şi metodele formelor în ansamblul lor. Nu vom acoperi toate proprietăţile şi metodele existente, ci doar pe acelea care ni s-au părut mai importante.

Caracteristici generale ale formelor

Prima dintre caracteristicile care trebuie stabilite la o formă (ca şi la orice obiect inclus într-o formă) este numele acesteia, ce serveşte la identificarea formei. Proprietatea care dă numele unui obiect este **Name** (nume). La crearea unui obiect, sistemul îi atribuie acestuia un nume implicit, care însă poate fi schimbat de utilizator. De exemplu, la crearea unei forme noi, sistemul îi atribuie acesteia numele **Form1**.

Prin urmare, ori de câte ori dorim să ne referim la o formă, vom folosi numele ei. Este posibilă însă şi o referire relativă, cu ajutorul construcţiei `ThisForm` (această formă), atunci când referirea se face din interiorul unei proprietăţi sau metode a formei respective.

O altă caracteristică importantă a unei forme este tipul său, controlat de proprietatea **WindowType**. Această proprietate poate lua două valori, şi anume:

- *0 – Modeless*, caz în care forma va fi una normală, nemodală, ea putând preda controlul altor ferestre deschise pe ecran simultan, în vederea lucrului concurent cu mai multe ferestre;
- *1 – Modal*, când forma va fi una modală, adică nu va permite predarea controlului altor ferestre deschise pe ecran decât după închiderea ei.

Vom folosi tipul normal pentru majoritatea formelor utilizate la introducerea datelor. Tipul modal va fi întrebuinţat pentru acele ferestre care solicită de la utilizator un răspuns, deci nu permit trecerea mai departe până când nu s-a furnizat răspunsul aşteptat (de exemplu, ferestrele de afişare a diferitelor mesaje, care nu trebuie închise sau trecute în spatele altor ferestre până când utilizatorul nu a confirmat recepţia mesajului respectiv).

Ori de câte ori se rulează o formă de la care se aşteaptă un rezultat, forma respectivă va fi una modală. De exemplu, să presupunem că pentru completarea unui anumit câmp se lansează în execuţie o formă care permite utilizatorului selectarea elementului dorit dintr-o listă. După ce utilizatorul a făcut alegerea, elementul selectat este returnat programului apelant pentru a completa câmpul respectiv. Forma care permite selecţia elementului din listă trebuie să fie una modală, deoarece ea returnează programului un rezultat (elementul selectat).

Obs

Lansarea în execuţie din cadrul unei secvenţe de cod (o metodă a unui obiect, de exemplu) a unei forme modale determină oprirea rulării secvenţei de cod respective până la închiderea formei modale apelate.

Lansarea în execuţie dintr-o secvenţă de cod a unei forme nemodale (normale) nu determină oprirea rulării secvenţei de cod respective, ci permite continuarea rulării „în paralel" cu secvenţele de cod ataşate formei nemodale respective („rulare concurentă").

Ascunderea unei forme se face cu ajutorul proprietăţii **Visible** (vizibilă). Dacă această proprietate are valoarea `.T.`, forma va fi vizibilă pe ecran. În caz contrar, ea este ascunsă, adică este memorată doar în memoria internă a sistemului şi nu apare pe ecranul monitorului.

De exemplu, dacă vrem să ascundem o formă, vom executa comanda:

```
<nume formă>.Visible = .F.
```

Chiar dacă o formă este afişată pe ecran (vizibilă), accesul la ea poate fi oprit prin intermediul proprietăţii **Enabled** (disponibilă). Dacă această proprietate are valoarea logică *adevărat*, forma poate primi controlul, în caz contrar ea fiind indisponibilă pentru utilizator (orice încercare de a trece formei controlul va eşua).

Stabilirea aspectului exterior al formei

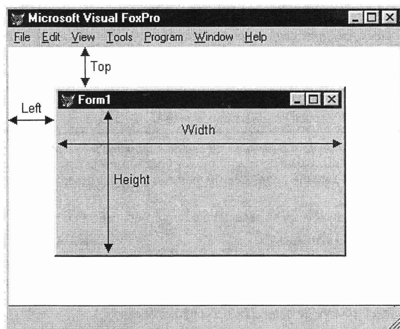
Primele detalii legate de aspectul exterior al unei forme sunt poziţia pe ecran şi dimensiunile formei. Aceste două caracteristici ale unei forme pot fi stabilite şi în mod interactiv, cu mouse-ul, şi prin specificarea explicită a coordonatelor şi dimensiunilor.

Poziţia pe ecran a unei forme este stabilită prin tragerea cu mouse-ul a zonei sale de titlu (clic simplu şi deplasarea cursorului mouse-ului, cu butonul stâng apăsat, în noua poziţie). Dimensiunile, orizontală şi verticală, ale formei se pot modifica interactiv prin tragerea cu mouse-ul a laturilor şi a colţurilor chenarului în poziţiile dorite.

Proprietăţile care stabilesc exact poziţia şi dimensiunile formei sunt prezentate în următoarea figură.

O proprietate specială a unei forme este **AutoCenter**, care, în cazul unei valori logice *adevărat* (`.T.`), determină poziţionarea automată a formei în centrul ecranului, fără a mai ţine cont de valorile proprietăţilor **Top** şi **Left**.

Titlul formei reprezintă textul care este afişat de sistem pe latura ei superioară. Acest titlu este determinat de textul atribuit proprietăţii **Caption** a formei.



Chenarul formei este stabilit de proprietatea **BorderStyle**. Aceasta poate lua următoarele valori:

- *0 – No border*, fără chenar;
- *1 – Fixed Single*, chenar simplu;
- *2 – Fixed Dialog*, chenar de tip fereastră de dialog;
- *3 – Sizable (Default)*, chenar dimensionabil.

Dintre toate aceste variante, numai ultima (care este şi varianta implicită) permite redimensionarea interactivă a formei de către utilizator (la rulare).

Culoarea de fundal a formei este stabilită prin intermediul proprietăţii **BackColor**, iar cea a cernelii cu proprietatea **ForeColor**. Această ultimă proprietate este utilă pentru stabilirea culorii cu care este afişat textul pe formă, afişare realizată prin intermediul comenzilor din metodele formei.

Aceeaşi observaţie este valabilă şi în cazul fontului folosit pentru afişarea textului direct pe formă, prin comenzile din metode. Proprietăţile de tip **Font...** (**FontName**, **FontSize** etc.) sunt explicate pe larg la obiectele de tip text informativ.

Câteva caracteristici legate de comportament

Am inclus în această categorie câteva dintre proprietăţile clasice ale unei ferestre, şi anume: posibilităţile de închidere, de mutare în altă poziţie, de minimizare şi de maximizare.

O formă poate fi închisă dacă proprietatea sa **Closable** are valoarea logică *adevărat* (`.T.`); în caz contrar (`.F.`), închiderea formei nu este permisă. Mutarea unei forme pe ecran este posibilă dacă proprietatea **Movable** (mutabilă) are valoarea logică *adevărat*.

Posibilitatea de maximizare a ferestrei este dată de prezenţa butonului de maximizare în partea din dreapta-sus a formei. Acest buton este afişat în poziţia respectivă dacă proprietatea **MaxButton** are valoarea `.T.`. În mod analog funcţionează şi minimizarea, proprietatea corespunzătoare a formei fiind **MinButton**.

În mod normal, o formă are în partea din stânga-sus o pictogramă, la acţionarea căreia este afişat un meniu cu comenzi de control al formei. În acest meniu se află comenzi precum **Move** (mutare), **Minimize** (minimizare), **Maximize** (maximizare) sau **Close** (închidere), cu ajutorul cărora se execută operaţiile respective asupra formei. Accesul la acest meniu este controlat de proprietatea **ControlBox**. În cazul unei valori *adevărat* a acestei proprietăţi, meniul de control al formei este disponibil pentru utilizator.

Starea de afişare (la dimensiuni normale, maximizată sau minimizată) este stabilită prin proprietatea **WindowState**. Această proprietate poate lua trei valori: *0 – Normal*, *1 – Minimized* (minimizată) şi *2 – Maximized* (maximizată), corespunzătoare celor trei stări amintite.

Exemplu

*De exemplu, dacă se doreşte ca o formă să ocupe tot ecranul la afişare, se stabileşte pentru proprietatea **WindowState** valoarea 2 (maximizată).*

Mediul de date al formei

Unul dintre principalele elemente legate de configurarea mediului de date în care lucrează o formă este ansamblul zonelor de lucru şi tabelele deschise în ele. Ori de câte ori în interiorul unei forme sunt modificate date dintr-o tabelă, aceasta trebuie (evident) să fi fost deja deschisă. Sistemul permite ca la rularea unei forme să fie deschise automat (fără comenzi speciale incluse în vreo metodă a formei) anumite tabele şi, eventual, să fie stabilite legături între ele. Pentru aceasta, la construirea formei în Constructorul de forme, trebuie specificate tabelele ce urmează a fi deschise automat.

Pentru a face acest lucru, se deschide fereastra mediului de date a Constructorului de forme. Dacă această fereastră nu conţine tabele (nu au fost specificate anterior), sistemul deschide automat o fereastră de dialog în care se solicită proiectantului tabelele deschise automat la rularea formei (a se vedea figura următoare).

La mediul de date al formei se pot adăuga mai multe tabele, eventual incluse în baze de date. Dacă între tabelele adăugate la mediul de date al formei există relaţii permanente (memorate în bazele de date), acestea vor fi restabilite automat.

O dată incluse în mediul de date al unei forme, tabelele vor fi deschise automat în sesiunea de date a formei respective, indiferent dacă aceasta este cea implicită a sistemului sau este una privată, proprie formei. Dacă, înainte de rularea formei, tabelele incluse în mediul ei de date au fost deja deschise în sesiunea de date respectivă, va fi generat un mesaj de eroare, deoarece sistemul încearcă să le redeschidă (nu se presupune implicit prezenţa clauzei AGAIN a comenzii USE).

Definirea unor tabele în mediul de date al unei forme este recomandată atunci când numele şi poziţia acestora sunt fixe, adică sunt cunoscute apriori, la construirea formei de către proiectant.

Dacă însă numele sau locul tabelelor folosite în interiorul formei sunt variabile, adică sunt cunoscute doar la rularea formei, atunci trebuie folosită o metodă manuală de deschidere în sesiunea de date a formei. Vom folosi pentru aceasta o serie de evenimente specifice mediului formei, care vor fi discutate în cele ce urmează.

Mediul de date al unei forme este considerat un obiect, prin urmare are asociate proprietăţi şi metode. Una dintre metode este **OpenTables** (deschidere tabele), ce poate fi folosită pentru deschiderea tabelelor necesare. De asemenea, pentru deschiderea tabelelor s-ar putea folosi şi metoda **Load** (încărcare), care este apelată după **OpenTables**, dar înaintea metodelor **Init** (de iniţializare) ale formei şi ale obiectelor componente ale acesteia.

Condiţia enunţată anterior este necesar a fi respectată deoarece în momentul execuţiei metodelor **Init** ale obiectelor formei, tabelele trebuie să fie deja deschise, pentru că unele obiecte se iniţializează cu valori din aceste tabele.

Închiderea tabelelor se poate face în metoda **CloseTables** (închidere tabele) a mediului de date al formei, alte variante fiind metodele **Unload** (descărcare) sau **Destroy** (distrugere) ale formei.

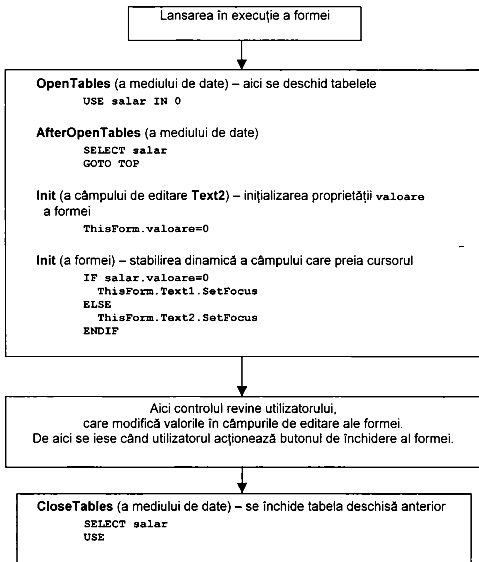
O altă operaţie legată de mediul de date al formei este şi iniţializarea variabilelor folosite în formă (ca variabile independente sau ca proprietăţi ale formei). Acest lucru se poate face în metoda **Init** a formei sau a obiectelor de interfaţă componente (fiecare obiect are asociată o metodă **Init**).

Exemplu

*Să presupunem că avem o formă care conţine două câmpuri de editare, numite **Text1** şi **Text2**. În primul dintre ele se editează valoarea câmpului* `valoare` *al*

tabelei **`salar`** *(`salar.valoare`), iar în cel de-al doilea valoarea unei proprietăți a formei (o variabilă) numită tot* **`valoare`** *(`ThisForm.valoare`).*

În următoarea schemă este prezentată simplificat succesiunea de comenzi asociate diferitelor metode ale formei, în ordinea apelării lor de către sistem.



Observăm locul şi ordinea de apelare a metodelor **Init** (ale formei şi ale obiectelor). Mai întâi sunt apelate metodele **Init** asociate obiectelor de interfaţă ale formei şi abia apoi cea asociată formei în ansamblu.

În metoda **Init** a obiectului **Text2** a fost iniţializată valoarea proprietăţii `valoare` a formei, iar în metoda **Init** a formei a fost stabilit obiectul care va primi controlul (cursorul), adică obiectul care va deveni ţinta intrărilor.

Principalele metode ale mediului de date care se pot folosi pentru gestiunea tabelelor dintr-o formă sunt următoarele:

- **BeforeOpenTables** – înainte de deschiderea tabelelor – aici se pot stabili dinamic numele şi locaţia tabelelor ce urmează a fi deschise;
- **OpenTables** – deschiderea tabelelor – aici se includ comenzile pentru deschiderea efectivă a tabelelor;
- **AfterOpenTables** – după deschiderea tabelelor – în această metodă se poate selecta tabela curentă şi înregistrările curente;
- **BeforeCloseTables** – înainte de închiderea tabelelor – se pot efectua calcule statistice asupra tabelelor modificate în formă;
- **CloseTables** – închiderea efectivă a tabelelor – se închid efectiv tabelele formei;
- **AfterCloseTables** – după închiderea tabelelor;

De asemenea, există şi o serie de proprietăţi ale mediului de date, dintre care amintim:

- **InitialSelectedAlias** – stabileşte tabela care va fi selectată iniţial;
- **AutoOpenTables** – determină, în cazul valorii logice `.T.`, deschiderea automată a tabelelor specificate în mediul de date al formei;
- **AutoCloseTables** – determină, în cazul valorii logice `.T.`, închiderea automată a tabelelor specificate în mediul de date al formei.

Sesiunea de date a formei

Sesiunea de date este o noţiune nouă în Visual FoxPro şi oferă posibilitatea de a avea un mediu de date specific fiecărei forme. Spre deosebire de versiunile FoxPro anterioare, pentru care configurarea mediului era aceeaşi indiferent de programul rulat (ecran de introducere a datelor, raport etc.), în Visual FoxPro se permite ca fiecare formă (raport etc.) să aibă propriul mediu de date, adică propriile configurări ale comenzilor `SET`, propriile zone de lucru în care pot fi deschise tabele, eventual cu relaţii stabilite între ele, propriile zone de memorie pentru variabile.

Def *O **sesiune de date** reprezintă o anumită configurație a parametrilor mediului SGBD (inclusiv a zonelor de lucru şi a variabilelor), specifică unui anumit element (formă, raport etc.).*

La nivelul sistemului Visual FoxPro există o **sesiune de date implicită**. O formă (sau un raport) poate lucra în sesiunea de date implicită sau poate avea propria sesiune de date (numită şi **privată**). Prima variantă se foloseşte atunci când, la crearea formei, se doreşte preluarea configurației implicite a mediului SGBD.

Sesiunea de date specifică formei se va folosi atunci când se doreşte o mai mare independență a formei față de mediu. În acest caz însă, este necesar ca toate configurările mediului să fie efectuate la construirea formei respective, inclusiv deschiderea bazelor de date, a tabelelor, definirea anumitor variabile etc.

O formă posedă o serie de proprietăți şi metode referitoare la sesiunea sa de date; despre câteva dintre acestea vom discuta în continuare. Prima este **DataSession**, care stabileşte tipul sesiunii de date a formei:

- *1 – Default Data Session*, când forma nu are o sesiune de date proprie, ci lucrează în sesiunea de date implicită a sistemului;
- *2 – Private Data Session*, caz în care forma are o sesiune de date proprie, privată.

Sesiunilor de date existente la un moment dat în sistem li se atribuie un număr, prin intermediul cărora ele sunt identificate. Numărul sesiunii de date a unei forme se obține cu ajutorul proprietății **DataSessionId**.

Proprietățile prezentate mai sus sunt des folosite în cazul definirii unor forme care permit lucrul cu mai multe instanțe ale lor. De exemplu, în cazul unei forme pentru introducerea facturilor emise de o societate comercială, dacă se doreşte ca în acelaşi timp să poată fi deschise pe ecran mai multe facturi (deci mai multe instanțe ale formei respective), se va declara pentru formă sesiune de date privată, urmând ca identificatorul sesiunii de date să fie folosit acolo unde se construiesc elemente proprii fiecărei instanțe a formei (de exemplu, la crearea unor tabele temporare, în al căror nume se poate introduce numărul sesiunii de date a formei pentru a se asigura astfel unicitatea). Această tehnică va fi prezentată într-unul dintre paragrafele următoare ale capitolului.

Câteva evenimente la nivel de formă

Există o serie de evenimente care sunt interceptate de o formă şi la care aceasta poate răspunde conform dorințelor proiectantului. Pentru aceasta, proiectantul trebuie să introducă anumite comenzi în metodele corespunzătoare ale formei. Alegerea metodelor în care se va introduce o anumită secvență de comenzi se face în funcție de momentul în care se doreşte lansarea lor în execuție. Dacă, de exemplu, se doreşte ca

la apăsarea unei anumite taste să se execute o anumită comandă, comanda va fi introdusă în metoda ataşată evenimentului de apăsare a unei taste, **KeyPress**.

Prin urmare, configurarea comportamentului unei forme în diferite situaţii presupune cunoaşterea de către proiectant a evenimentelor la care forma poate răspunde. Câteva dintre acestea sunt prezentate în continuare.

La crearea unei forme (la rulare) este apelată automat procedura de iniţializare a acesteia, adică metoda **Init**, iar la distrugerea formei (eliminarea sa de pe ecran şi din memorie) este executată metoda **Destroy**. Prin urmare, dacă dorim ca în unul dintre momente să fie executate anumite comenzi, acestea vor fi incluse în una dintre metode.

Metoda **Init** este şi cea care preia parametrii de rulare ai formei, transmişi acesteia de programul apelant. Comanda de execuţie a unei forme este asemănătoare cu cea de execuţie a unui program, adică:

```
DO FORM <nume formă> WITH <listă parametri>
```

În această comandă, lista de parametri reprezintă o serie de variabile sau valori de diferite tipuri transmise formei la rulare. Forma preia aceşti parametri prin metoda **Init**, a cărei primă linie trebuie să fie de tipul:

```
PARAMETERS <listă parametri formali>
```

(Evident, linia PARAMETERS va lipsi dacă forma se apelează fără parametri exteriori).

Dacă argumentele preluate de formă în parametrii formali ai comenzii PARAMETERS sunt necesare şi în alte metode ale formei decât **Init**, atunci acestea trebuie memorate fie în variabile globale, fie în proprietăţi special ataşate formei în acest scop.

Transferul invers de parametri, de la formă la programul apelant, se poate realiza fie direct prin parametrii metodei **Init** (când transferul se face tot în această metodă – caz mai rar), fie cu ajutorul metodei **Unload** (despre care vom discuta mai departe).

O formă poate conţine mai multe obiecte de interfaţă cu utilizatorul, fiecare dintre acestea având ataşată propria metodă **Init**. Metoda **Init** a unui obiect de interfaţă este apelată atunci când se creează obiectul respectiv, la rularea formei. Precedenţa între metoda **Init** a formei şi metodele **Init** ale obiectelor sale componente este următoarea: mai întâi sunt executate metodele **Init** ale obiectelor şi abia apoi cea a formei.

Ţinând cont de observaţia de mai sus, se poate ajunge la o situaţie specială, şi anume aceea în care iniţializarea unui obiect necesită date care nu sunt disponibile decât după execuţia unor comenzi la nivel de formă (de exemplu, un obiect are nevoie de date dintr-o anumită tabelă, care încă nu a fost deschisă). Altfel spus, în metoda **Init** a unui obiect sunt necesare date care se obţin abia după rularea metodei **Init** a formei. În acest caz este folosită metoda **Load** (încărcare).

Metoda **Load** este apelată imediat după crearea formei, dar înainte de metoda **Init** (şi înainte de alte metode, precum cea de activare a formei, **Activate**, sau cea de preluare a controlului, adică **GotFocus**). În această metodă se pot introduce comenzi preliminare creării obiectelor conţinute de formă, precum cele de iniţializare a unor variabile, de deschidere a unor tabele etc.

Metoda opusă metodei **Load** este **Unload** (descărcare), care este apelată la distrugerea obiectului. Aceasta este ultima metodă apelată la distrugerea unei forme şi deci este apelată după toate celelalte metode asociate formei, cum ar fi de exemplu **Destroy** (distrugere).

Metoda **Unload** a unei forme este folosită şi pentru returnarea unor parametri programului apelant. Acesta este cazul unei forme folosite ca un modul independent pentru citirea unor date de la utilizator. Pentru citirea datelor respective se apelează forma, iar după terminarea introducerii datelor este necesară returnarea lor către programul apelant. Acest lucru s-ar putea realiza prin intermediul unei tabele temporare, dar şi direct, cu ajutorul metodei **Unload**.

Introducerea comenzii

```
RETURN <expresie>
```

în metoda **Unload** determină transmiterea către programul apelant a valorii rezultate prin evaluarea expresiei respective. Programul apelant trebuie să preia valoarea într-o variabilă, care trebuie specificată în instrucţiunea de apelare a formei:

```
DO FORM <nume formă> WITH <listă de parametri> TO <variabilă>
```

Rezumând, în comanda de apelare a unei forme se pot specifica o serie de parametri care să fie transmişi acesteia cu ajutorul clauzei `WITH`, urmând ca răspunsul formei să fie preluat în variabila clauzei `TO`. În ceea ce priveşte forma apelată, ea preia parametrii transmişi de programul apelant (clauza `WITH`) în metoda **Init** (comanda `PARAMETERS`) şi returnează spre programul apelant o valoare prin comanda `RETURN` în metoda **Unload**.

Observăm că de la forma apelată la programul apelant se poate transmite o singură valoare. Dacă se doreşte transmiterea mai multor valori, acestea se pot combina într-una singură, de exemplu de tip şir de caractere.

Exemplu

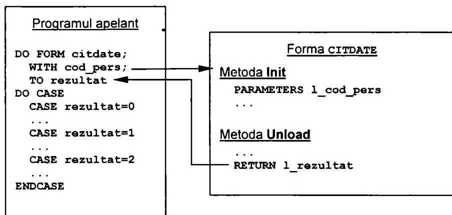
Să presupunem că dorim construirea unei forme în care să se citească datele referitoare la o anumită persoană. Codul persoanei este transmis formei ca parametru de către programul apelant, urmând ca forma să returneze un cod numeric indicând modul în care s-a terminat sesiunea de introducere a datelor persoanei respective:

0 – pentru terminarea cu bine;

1 – pentru renunțarea la introducerea datelor;

2 – pentru terminarea definitivă a lucrului cu programul respectiv.

În figura de mai jos este schițat modul în care este construită forma respectivă, din punct de vedere al apelului ei și al transferului de parametri între module.

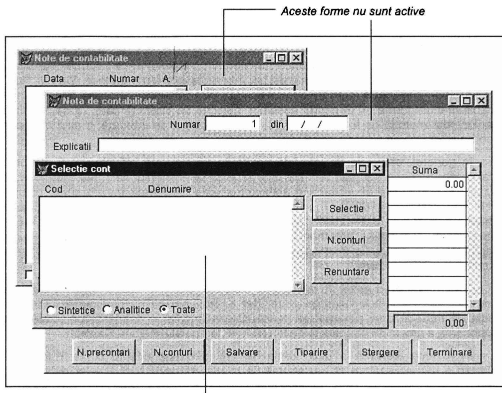


Definirea unei forme nu presupune și afișarea ei pe ecran; pentru afișare, este necesară apelarea unei metode speciale a formei, și anume **Show** (afișare). O dată apelată această metodă, forma definită anterior, dar invizibilă pentru utilizator, va deveni vizibilă.

Operația inversă, adică ascunderea unei forme, se face cu ajutorul metodei **Hide** (ascundere). O formă are și o proprietate specială care controlează starea sa de afișare. Această proprietate este **Visible** (vizibilă) și a fost prezentată într-un paragraf anterior.

Chiar dacă o formă este invizibilă, pe suprafața ei se pot afișa diferite texte sau obiecte de interfață, urmând ca acestea să apară pe ecran atunci când forma va deveni vizibilă.

La un moment dat, pe ecran pot fi afișate mai multe forme, însă dintre acestea una singură este activă, în sensul că reprezintă pentru utilizator ținta intrărilor (o tastă acționată de utilizator este preluată de forma respectivă și, în cadrul acesteia, de obiectul activ). Forma activă este afișată cu culori distincte (dacă nu au fost schimbate culorile implicite ale sistemului), pentru a pune în evidență această proprietate a sa.



Aceasta este forma activă

Activarea unei forme are loc atunci când utilizatorul execută un clic (cu mouse-ul) pe forma respectivă sau când este executată metoda **Show** (de afişare) a formei. Ori de câte ori un obiect al unei forme neactivate devine activ (primeşte controlul), este declanşat un eveniment **Activate** (activare) pentru forma respectivă.

Evenimentul de dezactivare, **Deactivate**, a unei forme are loc atunci când o formă activă transferă controlul altei forme, care devine activă.

Transferul controlului de la o formă la alta şi în cadrul aceleiaşi forme, de la un obiect la altul, este controlat de evenimentele **GotFocus** (preluare control) şi **LostFocus** (pierdere control). Atunci când utilizatorul execută un clic cu mouse-ul pe un obiect al formei active, obiectul care avea controlul va suporta un eveniment **LostFocus**, iar cel care primeşte controlul (cel pe care s-a acţionat cu mouse-ul) va suporta un eveniment **GotFocus**.

La nivel de forme, pierderea controlului de către o formă este semnalată de evenimentul **LostFocus**, iar primirea controlului de către o formă este semnalată de evenimentul **GotFocus**.

De asemenea, există posibilitatea transferării controlului de la o formă la alta sau de la un obiect la altul direct prin instrucţiuni introduse în diferite metode. De exemplu, apelul metodei **SetFocus** (predare control sau stabilire ţintă intrări) poate fi folosit pentru a stabili manual forma şi, în cadrul acesteia, obiectul de interfaţă care va deveni ţinta intrărilor. Altă metodă de a muta controlul de la un obiect de interfaţă la altul este prin valoarea returnată de metoda **Valid** (de validare).

În general, obiectele de interfaţă conţinute de o formă reflectă valori ale unor variabile de memorie sau câmpuri ale unor tabele. De exemplu, valoarea introdusă de utilizator într-un câmp de editare al unei forme este memorată într-un câmp al unei tabele sau într-o variabilă de memorie, deci acel câmp de editare reflectă valoarea câmpului tabelei sau a variabilei corespunzătoare. Dacă valoarea variabilei (câmp al tabelei) este însă modificată manual (prin instrucţiuni introduse în diferite metode) obiectul de control trebuie avertizat cu privire la această modificare, pentru a fi astfel actualizat cu noua valoare.

Ori de câte ori o astfel de valoare se modifică, sistemul reface starea controlului respectiv pentru a indica valoarea nouă, dar acest lucru nu se realizează totdeauna imediat. Pentru a forţa sistemul să reactualizeze imediat o formă şi obiectele sale componente astfel încât să reflecte noile valori, este apelată metoda **Refresh**.

În metoda **Refresh** a unei forme se pot introduce diferite comenzi care să realizeze o reîmprospătare de un anumit tip a unei forme şi a obiectelor de interfaţă componente.

Exemplu

În următoarea formă sunt citite de la utilizator două valori în variabilele a şi b. Câmpul de editare corespunzător variabilei c va indica suma variabilelor a şi b, fiind actualizat automat imediat ce utilizatorul modifică una dintre valori, a sau b.

Suma

a 2

b 3

c=a+b 5

*În forma de mai sus au fost definite cele trei câmpuri de editare asociate variabilelor a, b şi c. În metoda **Valid** a primelor două câmpuri a fost introdusă instrucţiunea*

```
Suma.Refresh
```

*pentru ca, după modificarea oricăreia dintre cele două valori, să fie apelată automat metoda **Refresh** a formei. În aceast a fost introdusă instrucţiunea*

```
c=a+b
```

pentru a reactualiza de fiecare dată valoarea lui c.

Obiecte de interfaţă ce pot fi incluse în forme

O formă reprezintă o fereastră cu ajutorul căreia utilizatorul îşi va preciza opţiunile, în care el va introduce parametrii doriţi. Pentru ca o formă să devină operaţională, trebuie ca ea să conţină obiecte de interfaţă, prin intermediul cărora se va realiza interacţiunea cu utilizatorul.

Există mai multe tipuri de obiecte de interfaţă, numărul acestora fiind practic nelimitat. Oricine poate proiecta propriul său tip de obiecte de interfaţă, în funcţie de modul în care doreşte ca acestea să apară pe ecran şi să interacţioneze cu utilizatorul. În timp, unele tipuri de obiecte de interfaţă s-au standardizat, cele mai importante fiind prezentate în continuare.

Pentru fiecare dintre aceste obiecte se va prezenta utilitatea şi modul de folosire, proprietăţile şi metodele specifice. Nu vor fi acoperite, desigur, toate proprietăţile şi metodele fiecărui obiect de interfaţă, ci numai acelea care ni s-au părut mai importante.

O serie de proprietăţi şi metode care sunt prezentate pentru un anumit obiect de interfaţă sunt aplicabile şi altor tipuri de obiecte. De exemplu, evenimentul **Click**, de acţionare cu mouse-ul asupra unui anumit obiect, are sens pentru majoritatea tipurilor de obiecte de interfaţă şi, mai mult, la nivel de formă în ansamblu. El este prezentat în detaliu la butoane, dar se aplică şi obiectelor grafice (linii, chenare, elipse sau imagini), câmpurilor de editare, comutatoarelor etc.

Textele informative (Label)

Prin texte informative am desemnat acele obiecte de interfaţă cu utilizatorul cu ajutorul cărora sunt afişate în cadrul unei forme diferite texte. Termenul englezesc pentru acest obiect este „Label", tradus exact „etichetă".

Definirea unui text informativ debutează prin acţionarea butonului **A** de pe bara utilitară a obiectelor de interfaţă. O dată acţionat acest buton, se vor stabili pe formă, cu ajutorul mouse-ului, poziţia şi dimensiunea noului obiect.

Obiectul de tip text informativ reprezintă o zonă a unei forme în care este afişat textul respectiv. Dimensiunea obiectului reprezintă, de fapt, dimensiunea zonei respective. Prin urmare, proprietăţile **Top**, **Left**, **Height** şi **Width** stabilesc poziţia pe verticală şi orizontală şi, respectiv, dimensiunile pe verticală şi orizontală ale zonei rezervate afişării textului, dimensiuni exprimate în pixeli şi calculate relativ la colţul din stânga-sus al formei.

Principala proprietate a unui text informativ este **Caption**, ea desemnând textul afişat în interiorul obiectului (aceasta este, evident, de tip şir de caractere).

Alinierea textului în cadrul obiectului este stabilită prin proprietatea **Alignment** (aliniere), care poate lua următoarele valori:

- *0 – Left (Default)* – aliniere la stânga (variantă implicită);
- *1 – Right* – aliniere la dreapta;
- *2 – Center* – aliniere în centru.

Alte caracteristici importante ale unui obiect de acest tip sunt cele legate de fontul folosit pentru afişare, adică:

- numele fontului sau al setului de caractere folosit – de exemplu *Arial*, *Courier New* etc. Proprietatea folosită în acest caz este **FontName** (nume font);
- dimensiunea fontului – dată de proprietatea **FontSize**, specifică dimensiunea în pixeli a caracterelor. Valoarea introdusă trebuie să fie de tip numeric;
- **îngroşarea fontului** – specifică, în cazul unei valori logice *adevărat*, afişarea caracterelor **cu linie îngroşată**. Proprietatea corespunzătoare este **FontBold**;
- *înclinarea fontului* – specifică, în cazul unei valori logice *adevărat*, afişarea caracterelor înclinate spre dreapta. Proprietatea corespunzătoare este **FontItalic**;
- sublinierea fontului – specifică, în cazul unei valori logice *adevărat*, sublinierea cu o linie simplă a caracterelor. Proprietatea corespunzătoare este **FontUnderline**;
- **FontCondense** şi **FontExtend** sunt două proprietăţi care indică prezenţa sau absenţa condensării, respectiv extinderii pe orizontală a textului afişat;

- alte efecte speciale sunt date de proprietățile **FontOutline** (text cu linie exterioară delimitatoare), **FontShadow** (text cu umbră) și **FontStrikethrough** (~~text tăiat cu o linie~~);

Culorile folosite pentru afișarea textului sunt specificate prin intermediul proprietăților **ForeColor** (culoarea de trasare) și **BackColor** (culoarea de fundal a întregii zone ocupate de obiect).

O proprietate specială a unui obiect de tip text informativ este **WordWrap**. Aceasta stabilește dacă un text mai lung, care depășește dimensiunea orizontală a zonei de afișare a obiectului, va fi împărțit pe rânduri succesive.

fără **WordWrap** (`.F.`) →	este un text mai lung care nu încape pe un singu
cu **WordWrap** (`.F.`) →	Acesta este un text mai lung care nu încape pe un singur rând

Imagini, linii și chenare

Pe lângă texte, pe suprafața unei forme pot fi afișate și alte tipuri de obiecte, grafice sau semigrafice. O primă categorie de astfel de obiecte sunt liniile, care pot apărea într-o mulțime de formate.

Butonul de pe bara utilitară a obiectelor de interfață care trebuie acționat pentru trasarea pe formă a unei linii este .

Principalele proprietăți ale unui obiect de tip linie sunt:

- culoarea de trasare a liniei – specificată prin intermediul proprietății **BorderColor**;
- stilul liniei – stabilit de proprietatea **BorderStyle**, care poate lua următoarele valori:
  - *0 – Transparent* (linia nu se vede);
  - *1 – Solid (Default)* (valoarea implicită), caz în care linia este continuă, trasată cu culoarea dată de **BorderColor** (————);
  - *2 – Dash*, pentru linie întreruptă (– – – – – – –);

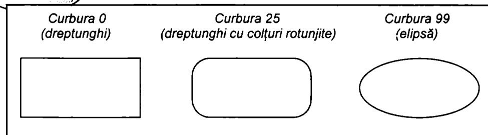
- *3 – Dot*, pentru linie punctată (............................);
- *4 – Dash – Dot*, pentru linie urmată de punct (–·–·–·–·–);
- *5 – Dash – Dot – Dot*, pentru linie urmată de două puncte (–··–··–··–··);
- *6 – Inside Solid*, pentru linie continuă în interior (————).

* grosimea liniei, care este stabilită în puncte, prin intermediul proprietăţii **BorderWidth**:
  - grosime de 1 punct (————);
  - grosime de 3 puncte (▬▬▬▬).
* modul de trasare în cadrul zonei de trasare (dat de proprietatea **LineSlant**). De fapt, trasarea unui obiect de tip linie se rezumă la trasarea unei zone dreptunghiulare în interiorul căreia este desenată linia:
  - începând din colţul din stânga-sus spre cel din dreapta-jos, pentru cazul în care proprietatea **LineSlant** are valoarea \;
  - începând din colţul din stânga-jos spre cel din dreapta-sus, pentru cazul în care proprietatea **LineSlant** are valoarea /.

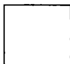
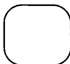
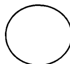
O altă categorie de obiecte grafice ce pot spori lizibilitatea unei forme sunt chenarele, cu colţuri drepte sau rotunjite, cercurile şi elipsele. Acestea sunt trasate ca urmare a acţionării butonului de pe bara utilitară a obiectelor de interfaţă.

Dreptunghiurile, pătratele, elipsele şi cercurile sunt realizate prin intermediul aceluiaşi obiect. De la dreptunghi la elipsă se ajunge prin modificarea gradului de rotunjire a colţurilor, începând de la 0 şi până la 99.

Exemplu



Dacă se porneşte de la un pătrat, prin rotunjirea colţurilor sale se ajunge la cerc.

Proprietatea care stabileşte gradul de rotunjire a colţurilor unui astfel de obiect este **Curvature** (curbură).

Pe lângă linii, chenare, elipse şi cercuri, pe suprafaţa unei forme mai pot fi afişate diferite imagini, care însă trebuie să fi fost create anterior cu un program de editare grafică.

Introducerea unei imagini într-o formă debutează prin acţionarea butonului de pe bara utilitară a obiectelor de interfaţă. Urmează trasarea cu mouse-ul a zonei de pe suprafaţa formei în care va fi afişată imaginea respectivă.

Fişierul din care se preia imaginea (care trebuie să fi fost creat anterior) este specificat cu ajutorul proprietăţii **Picture**. El poate fi selectat în mod interactiv. Pentru aceasta, se selectează obiectul imagine din cadrul formei (prin clic pe el), apoi, din lista proprietăţilor a ferestrei de proprietăţi a Constructorului de forme, se selectează proprietatea **Picture**. Urmează acţionarea butonului din dreapta câmpului de editare a valorilor proprietăţilor şi se deschide fereastra din care se poate alege fişierul dorit.

Imaginea poate fi încadrată de un chenar trasat cu linie simplă, dacă proprietatea **BorderStyle** are valoarea *1 – Fixed Single*, sau poate fi neîncadrată de chenar atunci când proprietatea respectivă are valoarea *0 – None (Default)*.

Deseori, dimensiunea imaginii memorate în fişierul sursă nu corespunde exact dimensiunii zonei rezervate de proiectant în formă. Proprietatea **Stretch** indică modul în care imaginea este încadrată în zona de formă rezervată ei, şi anume:

- *0 – Clip (Default)* – imaginea va fi afişată la dimensiunile originale. Dacă imaginea originală este mai mare decât zona de pe formă rezervată ei, atunci va fi afişată numai o parte a imaginii, cea care încape în zona respectivă;
- *1 – Isometric* – imaginea va fi micşorată sau mărită în funcţie de zona din formă care îi este rezervată, păstrându-se însă proporţiile originale (raportul între înălţime şi lăţime, fără distorsionare);
- *2 – Stretch* – imaginea va fi mărită sau micşorată şi, eventual, distorsionată, astfel încât să ocupe întreaga zonă din formă rezervată ei.

Obiectele de interfaţă prezentate anterior nu sunt unele simple, folosite doar pentru afişarea diverselor informaţii. Ele pot reacţiona la diferite evenimente, cum ar fi de exemplu un clic cu mouse-ul – **Click** – sau mutarea manuală de către utilizator a obiectului respectiv în altă poziţie (tragerea sa cu mouse-ul) – **Drag**.

În acest fel, un obiect de tip imagine poate deveni un buton pe care utilizatorul îl poate acţiona, iar un chenar poate deveni un instrument de delimitare interactivă a unei anumite zone dintr-o formă.

Câmpurile de editare (Text Box)

Câmpurile de editare reprezintă unele dintre cele mai folosite obiecte de interfaţă din programele de introducere a datelor, datorită faptului că ele permit introducerea de valori de diferite tipuri, precum numere, şiruri de caractere, date calendaristice etc.

Exemplu

Exemple de câmpuri de editare sunt prezentate în următoarea formă:

Nota de contabilitate

Numar 1 din / /

Explicatii

Zi	Explicatii	Debit	Credit	Suma
0				0.00

Anulata Nota introdusa manual Total 0.00

N.precontari N.conturi Salvare Tiparire Stergere Terminare

Definirea unui câmp de editare se realizează prin acționarea butonului  de pe bara utilitară a obiectelor de interfață, urmată de trasarea cu mouse-ul a zonei din formă ocupate de câmp. O dată definit câmpul respectiv, urmează parametrizarea sa, adică specificarea proprietăților şi a conținutului metodelor asociate lui.

Caracteristici generale ale câmpurilor de editare

Zona din formă rezervată câmpului de editare poate fi specificată exact prin intermediul proprietăților **Top**, **Left**, **Height** şi **Width**. Acestea indică distanța față de marginea de sus şi față de latura stângă a formei, înălțimea şi respectiv lățimea câmpului de editare.

Relativ la înălțimea câmpului de editare, există o proprietate specială numită **IntegralHeight**. Dacă are valoarea *adevărat*, ea determină stabilirea automată a înălțimii obiectului la o valoare calculată astfel încât textul din câmpul de editare să încapă complet în câmp – înălțimea nu este nici mai mare, nici mai mică decât este necesar.

Una dintre caracteristicile importante ce trebuie precizate la definirea unui câmp de editare este sursa de date a obiectului de interfață, dată de proprietatea **ControlSource**. Aceasta reprezintă o variabilă sau un câmp al unei tabele în care este memorată valoarea introdusă de utilizator în câmpul de editare. De fapt, câmpul de editare nu reprezintă decât forma exterioară prin care sunt reprezentate şi modificate variabila sau câmpul respectiv.

Alinierea textului în cadrul câmpului de editare este dată de proprietatea **Alignment**. Aceasta poate lua următoarele valori:

- *0 – Left* – pentru aliniere la stânga;
- *1 – Right* – pentru aliniere la dreapta;
- *2 – Center* – pentru aliniere în centru;
- *3 – Automatic (Default)* – caz în care alinierea textului este lăsată în seama sistemului. Pentru numere alinierea va fi la dreapta, iar pentru datele de celelalte tipuri alinierea va fi la stânga. Un caz special se întâlneşte la câmpurile de editare încorporate în grile, când alinierea este stabilită la nivel de coloană.

Margin reprezintă o proprietate care stabileşte distanța suplimentară dintre textul conținut în câmpul de editare şi marginea lui. Acest lucru este util pentru lizibilitatea câmpului.

Margine 0 | Margine 2 | Margine 4

Două proprietăţi referitoare la formatarea la citire şi la afişare a textului introdus în câmpul de editare sunt **Format** şi **Input Mask** (mască pentru introducere). Pentru cunoscătorii variantelor mai vechi ale SGBD, aceste proprietăţi sunt analoage clauzelor FUNCTION şi respectiv PICTURE ale comenzilor @...GET clasice.

Prin urmare, proprietatea **Format** conţine coduri cu diferite semnificaţii (numite coduri FUNCTION), prezentate în tabelele de mai jos. Codurile de acest tip se aplică asupra textului introdus sau afişat în câmpului de editare în ansamblul său, spre deosebire de codurile specificate pentru proprietatea **InputMask** (coduri PICTURE), care se aplică numai caracterului de pe poziţia corespunzătoare din câmpul de editare.

Semnificaţia codurilor FUNCTION (proprietatea Format)

Cod	Semnificaţie
!	Este folosit pentru valorile de tip şir de caractere, transformând toate literele în majuscule
$	Pentru valorile numerice, afişează simbolul monetar
^	Afişează valorile numerice în formatul ştiinţific
A	Permite numai litere, fără spaţii sau semne de punctuaţie
D	Determină afişarea datelor calendaristice în formatul stabilit de SET DATE
E	Editează valorile de tip dată calendaristică în formatul englezesc
K	Selectează textul câmpului atunci când controlul devine ţinta intrărilor
L	Afişează zerouri la sfârşitul valorilor numerice (după punctul zecimal) în locul spaţiilor
R	Afişează macheta de introducere specificată în proprietatea **InputMask** (fără însă ca masca să influenţeze valoarea memorată în câmpul de editare)
T	Înlătură spaţiile din faţa şi de la sfârşitul datelor de tip şir de caractere
YS	Afişează datele calendaristice în formatul scurt stabilit în Windows
YL	Afişează datele calendaristice în formatul lung stabilit în Windows

Semnificația codurilor PICTURE (proprietatea InputMask)

Cod	Semnificație
X	Permite introducerea oricărui caracter
9	Permite introducerea cifrelor şi a semnelor (cum ar fi minus, în cazul numerelor negative)
#	Permite introducerea cifrelor, a spaţiilor şi a semnelor
$	Afişează simbolul monetar (stabilit prin comanda SET CURRENCY)
$$	Afişează simbolul monetar mobil
*	În stânga valorilor sunt afişate asteriscuri
.	Indică poziţia punctului zecimal
,	Separă cifrele la stânga punctului zecimal (de exemplu, cifra miilor de cifra sutelor)

Datele calendaristice pot face, de asemenea, obiectul câmpurilor de editare. Atunci când variabila sau câmpul tabelei care se află în spatele câmpului de editare este de tip dată calendaristică, pentru datele introduse în acest câmp pot fi specificate câteva proprietăţi specifice datelor calendaristice. Două dintre cele mai importante sunt **DateFormat** (formatul datei calendaristice) şi **Century** (secolul).

Prima dintre proprietăţile amintite poate lua mai multe valori. Valoarea *0 – Default* indică faptul că formatul datei calendaristice introduse în câmpul de editare va fi dat de comanda SET DATE (a se vedea tipul de date „dată calendaristică"). Dacă se doreşte precizarea explicită a unui format se pot folosi celelalte valori (*1 – American*, *2 – ANSI*, *3 – British*...).

Proprietatea **Century** stabileşte prezenţa sau lipsa în formatul datei a celor două cifre referitoare la secol. Ea poate lua trei valori, şi anume:

- *0 – Off* – caz în care cifrele referitoare la secol vor lipsi din dată, adică anul va fi format din două cifre, secolul 20 fiind subînţeles;
- *1 – On* – când aceste cifre vor fi prezente în dată, adică anul va avea alocate patru cifre;
- *2 – Default* – când secolul este dat de comanda SET CENTURY (a se vedea tipul dată calendaristică în capitolul „Tipuri de date").

Culorile cu care este afişat textul în câmpul de editare sunt stabilite pr intermediul următoarelor proprietăţi:

- **BackColor** şi **ForeColor**, pentru culoarea de fond şi, respectiv, pentr cerneală;
- **SelectedBackColor** şi **SelectedForeColor**, folosite la stabilirea culorilor d fond şi de cerneală pentru textul selectat în cadrul câmpului de editare;
- **DisabledBackColor** şi **DisabledForeColor**, pentru culorile de fond şi d cerneală, în cazul în care câmpul de editare este dezactivat (proprietatea **Enabled** are valoarea *fals*).

Fontul este stabilit prin intermediul proprietăţilor de tip **Font...** (a se vedea obiectul de tip text informativ, unde aceste proprietăţi sunt prezentate pe larg).

Câmpul de editare este accesibil atunci când proprietatea **Enabled** (accesibil) are valoarea *adevărat* (`.T.`); în caz contrar (valoarea *fals*, `.F.`), el nu este accesibil. În acest din urmă caz, controlul va fi desenat cu culori diferite (a se vedea observaţia de mai sus), pentru a indica starea de inaccesibilitate.

Proprietatea **Enabled** este folosită în cazurile în care un anumit câmp de editare nu are sens pentru setul de date introdus în cadrul formei.

Exemplu

De exemplu, numele soţului sau al soţiei şi data căsătoriei nu au sens pentru persoanele necăsătorite:

Nume si prenume sot (sotie)
Casatorit(a)
Data casatoriei

Prin urmare, câmpurile respective sunt dezactivate la deselectarea comutatorului care corespunde stării civile şi devin din nou disponibile la selectarea comutatorului.

Această proprietate poate fi modificată şi în mod dinamic, în timpul rulării unei forme, cu ocazia unui anumit eveniment. În cazul formei de mai sus, la validarea comutatorului pentru căsătorit sau necăsătorit (metoda **Valid** a acestuia) se poate

introduce comanda de dezactivare a câmpurilor referitoare la datele partenerului de viaţă:

```
<câmp>.Enabled = .F.
```

Un câmp de editare (ca şi alte tipuri de obiecte de interfaţă) ar putea fi folosit numai pentru afişarea valorii variabilei sau a câmpului tabelei pe care o reprezintă, fără a oferi însă utilizatorului posibilitatea modificării acestei valori. Pentru a stabili acest comportament, se atribuie proprietăţii **ReadOnly** (doar pentru citire) valoarea *adevărat*, `.T.` Indisponibilizarea unui obiect de interfaţă cu ajutorul proprietăţii **Enabled** diferă de starea „doar pentru citire", stabilită prin intermediul proprietăţii **ReadOnly**, prin faptul că, în ultimul caz, obiectul poate deveni ţinta intrărilor (în primul caz nu), fără a afecta însă valoarea variabilei asociate.

Vizibilitatea unui câmp de editare (sau a altui tip de obiect de interfaţă) este stabilită prin proprietatea **Visible** (vizibil). Dacă aceasta are valoarea `.T.` obiectul va fi vizibil în formă, iar în caz contrar va fi invizibil, adică ascuns utilizatorului (chiar dacă el este acolo şi răspunde la evenimente).

Această proprietate poate fi folosită, de exemplu, în cazul a două obiecte suprapuse, care vor fi afişate sau nu în funcţie de îndeplinirea unei anumite condiţii.

Exemplu

De exemplu, dacă se citesc date referitoare la clienţii unei unităţi economice, setul de date solicitate utilizatorului depinde de tipul clientului, persoană fizică sau juridică. Obiectele corespunzătoare celor două cazuri ar putea fi suprapuse în aceeaşi zonă a formei şi afişate (`Visible=.T.`) numai în cazul îndeplinirii sau neîndeplinirii condiţiei respective.

Lungimea maximă a textului unui câmp de editare, în caractere, poate fi stabilită prin intermediul proprietăţii **MaxLength**. Această proprietate se poate folosi atunci când se citesc o serie de date cu lungime variabilă (de exemplu, o adresă), care însă trebuie să fie încărcate ulterior într-un câmp al unei tabele de o anumită dimensiune (fixă, stabilită la proiectarea tabelei).

Faţă de vechile versiuni de FoxPro, în Visual FoxPro a fost introdus mecanismul de manipulare a valorilor nule (`.Null.`), cu ajutorul căruia se poate gestiona starea necompletată a unui câmp. În mod implicit, atunci când valoarea variabilei asociate unui câmp de editare are valoarea `.Null.`, în câmpul respectiv este afişat textul *.Null.* Dacă se doreşte personalizarea mesajului afişat în cazul unei valori nule a variabilei asociate, se poate folosi proprietatea **NullDisplay**, care permite specificarea textului care să fie afişat când valoarea din câmp este nulă.

Exemplu

De exemplu, necompletarea unei date poate fi semnalată direct în cadrul câmpului, printr-un mesaj de tipul „Aici introduceţi data...".

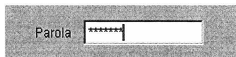
SelectOnEntry (selectare la intrare) este o proprietate a unui câmp de editare c poate lua doar valori logice. În cazul valorii *adevărat*, conținutul câmpului de editare va selectat automat atunci când câmpul devine ținta intrărilor, iar în cazul valorii *fals*, ace lucru nu se va întâmpla.

Proprietatea **SelectOnEntry** se folosește atunci când conținutul câmpului s schimbă la frecvent intrarea în câmp. Pentru că valoarea introdusă deja în câmp est selectată la intrare, o simplă apăsare a unei taste determină înlocuirea conținutulu respectiv cu caracterul corespunzător tastei apăsate. Prin urmare, nu este nevoie de acționare suplimentară de tastă pentru ștergerea conținutului existent, acesta fiind ște imediat ce se începe introducerea noului conținut.

Obs *Ultimele două proprietăți prezentate pot fi combinate pentru a se obține un efect special. Inițial, valoarea variabilei asociate câmpului de editare are valoarea* `.Null.` *și, prin intermediul proprietății* ***NullDisplay***, *se specifică un mesaj care să indice această stare a câmpului (necompletat). Conținutul inițial va fi imediat înlocuit cu noua valoare furnizată de utilizator, la prima apăsare de tastă, lucru posibil prin intermediul proprietății* ***SelectOnEntry***.

De obicei, un câmp de editare este însoțit și de un text informativ, care ofer utilizatorului detalii privind semnificația câmpului respectiv. În mod dinamic, atunci când un obiect de interfață are controlul (reprezintă ținta intrărilor utilizatorului), se poate afișa un mesaj în bara de stare a sistemului (din partea inferioară a ferestrei Visual FoxPro) mesaj specificat prin intermediul proprietății **StatusBarText**.

Introducerea codurilor secrete (a parolelor, de exemplu) trebuie efectuată fără ca un eventual observator care se află în spatele operatorului de la consolă să poată vedea codul respectiv. Proprietatea folosită pentru acest caz este **PasswordChar**, care permite specificarea unui caracter ce va fi afișat în locul caracterelor introduse de utilizator. Textul memorat în variabila asociată câmpului nu este afectat de această proprietate.



Metode ale câmpurilor de editare

Să trecem acum la studiul unor metode ale câmpurilor de editare, metode atașate anumitor evenimente. Vom începe cu **Init** și **Destroy**, cele două metode lansate în execuție la crearea, respectiv la distrugerea obiectului de interfață.

Există o metodă **Init** şi la nivel de formă, care este însă lansată în execuţie după metodele **Init** ale obiectelor de interfaţă. De asemenea, obiectele care conţin la rândul lor alte obiecte (de exemplu, grilele sau paginile alternative) au ataşate metode **Init**, care sunt lansate în execuţie după metodele **Init** ale obiectelor componente.

Metoda **Init** a unui obiect poate returna o valoare logică. Dacă această valoare este *adevărat*, obiectul va fi creat, iar în caz contrar se va renunţa la crearea sa. Metoda **Init** a unui obiect poate fi astfel folosită pentru realizarea unor teste (existenţa unor variabile, fişiere, tabele etc.), în urma cărora să se decidă dacă obiectul respectiv de interfaţă are sens.

Metoda **Destroy** este apelată la distrugerea obiectului de interfaţă. În cazul obiectelor compuse (care conţin alte obiecte), mai întâi este apelată metoda **Destroy** asociată obiectului compus şi apoi metodele asociate obiectelor de interfaţă componente.

Obiectul de interfaţă activ. Transferul ţintei intrărilor de la un obiect de interfaţă la altul

Def

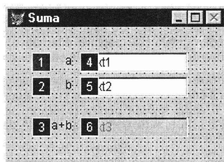
La un moment dat, în cadrul unei forme există un singur obiect de interfaţă în care utilizatorul introduce date, obiectul respectiv numindu-se ***activ****. Se spune că obiectul activ reprezintă* ***„ţinta intrărilor utilizatorului"****.*

Dacă se doreşte introducerea de date în alt obiect de interfaţă decât cel activ, este necesară trecerea controlului (cursorului, în cazul câmpurilor de editare) la noul obiect; abia apoi se pot scrie datele dorite. Schimbarea obiectului activ este echivalentă cu mutarea ţintei intrărilor de la un obiect la altul.

În mediul Windows, tasta consacrată pentru schimbarea obiectului activ (a ţintei intrărilor) este Tab (sau Ctrl+Tab). În cadrul unei forme pot fi definite mai multe obiecte, care, din punctul de vedere al ţintei intrărilor, sunt plasate într-o listă specială care dă ordinea lor de parcurgere. Prin urmare, există o ordine de acces (cu tasta Tab) la obiectele de interfaţă ale unei forme, ordine care poate fi însă modificată de utilizator după dorinţă.

Modificarea ordinii în care sunt parcurse obiectele de interfaţă ale unei forme se face astfel:

- din meniul **View** al sistemului se alege opţiunea **Tab Order** (ordinea de parcurgere cu tasta Tab), ocazie cu care pentru fiecare obiect de interfaţă în parte este afişat un mic pătrăţel în care apare numărul său de ordine în cadrul listei de parcurgere;



- urmează execuția unui clic cu mouse-ul pe fiecare pătrățel asociat obiectelor de interfață, în ordinea de parcurgere dorită. La executarea fiecărui clic, numerele de ordine din pătrățele se modifică, indicând noua ordine;
- operația se încheie atunci când se execută un clic în afara oricărui obiect de interfață sau când se alege iar opțiunea **Tab Order** a meniului **View**.

Se poate ca un obiect să fie exclus din lista de parcurgere, aceasta însemnând că el nu va mai fi parcurs la acționarea tastei Tab (și va fi accesibil doar cu mouse-ul). Pentru aceasta, se atribuie proprietății **TabStop** valoarea *fals* (în cazul *adevărat*, obiectul este inclus în lista de parcurgere cu tasta Tab).

Poziția unui obiect de interfață în lista de parcurgere cu tasta Tab este dată de proprietatea **TabIndex**. Metoda prezentată mai sus este una automată de stabilire a valorilor proprietății **TabIndex** pentru obiectele de interfață ale unei forme.

Evenimentele care însoțesc schimbarea obiectului activ (sau mutarea țintei intrărilor de la un obiect la altul) sunt următoarele:

- **GotFocus** – tradus „obținere control", are loc atunci când obiectul de interfață devine activ (sau primește ținta intrărilor);
- **LostFocus** – tradus „pierdere control", are loc atunci când obiectul de interfață respectiv devine inactiv, adică pierde ținta intrărilor (aceasta este transferată altui obiect de interfață, din aceeași formă sau din alta);
- **SetFocus** – tradus „stabilire control", transferă controlul obiectului de interfață respectiv, acesta devenind activ.

Exemplu

Reglarea modului în care se face transferul controlului de la un obiect la altul este foarte importantă în cadrul proiectării unei forme. De exemplu, selectarea unui comutator poate determina dezactivarea anumitor obiecte (a se vedea exemplul anterior cu comutatorul Căsătorit*), operație care poate fi realizată în*

*interiorul metodei **LostFocus**, adică atunci când comutatorul cedează controlul următorului obiect de interfaţă.*

*De asemenea, în cadrul metodei **LostFocus** se poate apela direct metoda **SetFocus**, pentru a transfera controlul la un anumit obiect. Comanda folosită va fi de forma:*

```
<obiect>.SetFocus
```

Validarea conţinutului unui câmp de editare

Una dintre cele mai importante metode ataşate unui câmp de editare este **Valid** (validare); aceasta este folosită pentru testarea corectitudinii valorii introduse de utilizator în câmp. Metoda este apelată înainte de pierderea controlului de către câmpul de editare. În funcţie de valoarea returnată de metoda **Valid**, controlul va fi cedat sau nu unui alt obiect de interfaţă. Această valoare poate fi de două tipuri:

- de tip logic, caz în care valoarea *adevărat* permite transferarea ţintei intrărilor spre următorul obiect de interfaţă, iar valoarea *fals* determină păstrarea controlului de către obiectul curent;
- de tip numeric, când valoarea 0 indică invalidarea valorii introduse în câmp (controlul nu va fi transferat spre alt obiect de interfaţă), iar o valoare diferită de 0 indică obiectul de interfaţă spre care va fi cedat controlul (în sens direct, ca şi tasta Tab, pentru valori pozitive, şi în sens indirect, echivalent cu combinaţia Alt+Tab, pentru valori negative).

Prin această ultimă facilitate se realizează atât validarea valorii introduse în câmpul de editare, cât şi stabilirea unei ordini speciale de parcurgere a obiectelor de interfaţă.

Exemplu

*De exemplu, s-ar putea ca în cazul unei valori negative introduse în câmpul de editare, următorul obiect de interfaţă să nu mai aibă sens şi deci să nu mai fie necesară parcurgerea sa. În acest caz, este suficient ca în metoda **Valid** să se testeze semnul valorii din câmp şi, în cazul unei valori negative, metoda să returneze valoarea 2 (salt peste următorul obiect şi predarea controlului celui de-al doilea obiect de interfaţă în ordinea de parcurgere).*

Accesibilitatea obiectului de interfaţă

Un eveniment analog cu **Valid**, care apare însă înainte ca un obiect de interfaţă să devină ţinta intrărilor, este **When**. În metoda asociată acestui eveniment se introduc de obicei comenzi cu ajutorul cărora se testează respectarea diferitelor condiţii

necesare pentru ca obiectul respectiv să fie accesibil. Dacă metoda **When** returnează valoarea *adevărat*, obiectul va deveni ţinta intrărilor, iar în cazul unei valori *fals*, obiectul de interfaţă va fi sărit (ţinta intrărilor va deveni următorul obiect de interfaţă).

Metoda **When** este folosită ori de câte ori se doreşte ca un anumit obiect de interfaţă să preia controlul condiţionat, adică în funcţie de respectarea sau nerespectarea unei anumite condiţii.

Personalizarea comportamentului câmpului de editare la acţionarea tastaturii

Acţionarea unei taste este interceptată de obiectul de interfaţă prin intermediul evenimentului **KeyPress**. Metoda asociată acestui eveniment este apelată ori de câte ori utilizatorul acţionează o tastă sau o combinaţie de taste, folosind tastele adiţionale Shift, Ctrl şi Alt. Prima linie a metodei trebuie să fie una de tip PARAMETERS, care să specifice parametrii locali în care sunt preluaţi parametrii transmişi de sistem. Sintaxa primei linii este:

```
PARAMETERS <cod tastă>, <cod taste adiţionale>
```

În primul dintre parametri este depus de către sistem codul tastei apăsate, câteva dintre aceste coduri fiind prezentate în următorul tabel.

Tastă	Cod tastă			
	Singură	cu Shift	cu Ctrl	cu Alt
F1	28	84	94	104
F2	-1	85	95	105
F3	-2	86	96	106
F4	-3	87	97	107
F5	-4	88	98	108
F6	-5	89	99	109
F7	-6	90	100	110
F8	-7	91	101	111
F9	-8	92	102	112
F10	-9	93	103	113
F11	133	135	137	139
F12	134	136	138	140
1	49	33	—	120
2	50	64	—	121
3	51	35	—	122
4	52	36	—	123
5	53	37	—	124
6	54	94	—	125
7	55	38	—	126

Cel de-al doilea parametru indică dacă tasta principală acționată a fost însoțită d una dintre tastele adiționale Shift, Ctrl sau Alt. Valoarea returnată se obține prin însuma- rea următoarelor valori:

- 1 dacă a fost acționată tasta Shift și 0 în caz contrar;
- 2 dacă a fost acționată tasta Ctrl și 0 în caz contrar;
- 4 dacă a fost acționată tasta Alt și 0 în caz contrar;

Exemplu

Prin urmare, dacă au fost apăsate tastele Shift și Alt, valoarea acestui parametru va fi 1+0+4=5.

Valoarea 7 se obține prin însumarea 1+2+4, ce arată că au fost acționate toate cele trei taste adiționale.

Acționarea tastei Alt este echivalentă cu returnarea pentru acest parametru a unei valori mai mari sau egale cu 4. Acționarea tastei Ctrl este indicată de o valoare a acestui parametru în mulțimea {2,3,6,7}. Un număr impar indică acționarea tastei Shift.

Alte două evenimente care indică modificarea dinamică a valorii din câmpul d editare (și în general a obiectului de interfață) sunt **InteractiveChange** ș **ProgrammaticChange**. Primul dintre evenimente apare atunci când valoarea din câm este modificată de utilizator cu ajutorul tastaturii sau al mouse-ului, iar cel de-al doile atunci când valoarea din câmp este modificată prin program (o comandă din cadrul une metode).

Metoda **InteractiveChange** poate fi folosită, de exemplu, pentru actualizare imediată a unui alt obiect de interfață al formei, chiar în timpul introducerii datelor î obiectul curent.

Reîmprospătarea obiectului de interfață

Deseori, valoarea afișată în câmpul de editare (sau în alt obiect de interfață) r reflectă exact valoarea variabilei asociate acestuia. Un exemplu este acela câ valoarea variabilei este modificată prin program, caz în care este necesară reîmprospătare a obiectului de interfață, pentru ca acesta să reflecte exact no valoare.

Evenimentul de reîmprospătare a unui obiect este numit **Refresh**. Există asemenea eveniment și la nivel de formă. Reîmprospătarea unui obiect este însoțită apelarea metodei **Refresh**, care se poate folosi astfel pentru împrospătarea individua a fiecărui obiect în parte.

Selectarea textului în cadrul unui câmp de editare

La introducerea unui text în cadrul unui câmp de editare, utilizatorul dispune de o serie de facilităţi specifice unui editor de text, una dintre cele mai importante fiind selectarea unor porţiuni din text (însoţită eventual de ştergere, înlocuire, copiere sau mutare a porţiunii respective). Când un câmp de editare reprezintă ţinta intrărilor, deplasarea cursorului peste o porţiune de text menţinând tasta Shift apăsată determină selectarea porţiunii respective de text.

Pentru controlul operaţiei de selectare a unui text într-un câmp de editare proiectanţii sistemului Visual FoxPro au prevăzut trei proprietăţi specifice, şi anume **SelStart**, **SelLength** şi **SelText**. Prima dintre proprietăţi indică poziţia de la care începe porţiunea de text selectată, iar cea de-a doua lungimea, în caractere, a porţiunii respective. În fine, ultima proprietate, adică **SelText**, conţine textul selectat.

Valorile acestor proprietăţi se modifică în mod dinamic (pe măsură ce utilizatorul selectează textul). Ele pot fi modificate şi direct, prin comenzi introduse în metodele formei şi ale obiectelor acesteia, ceea ce oferă proiectanţilor posibilităţi deosebite de manipulare a textelor din câmpurile de editare.

Exemplu

De exemplu, introducând într-o metodă comanda:

```
<camp>.SelText = "text nou"
```

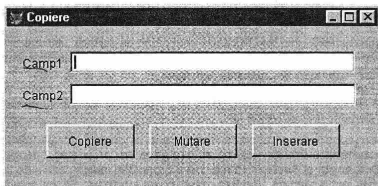
textul curent selectat în cadrul câmpului `<camp>` va fi înlocuit cu textul (şirul de caractere) `"text nou"`.

Selectarea caracterelor 3, 4 şi 5 din cadrul unui câmp de editare se poate face cu următoarea secvenţă de comenzi:

```
<camp>.SelStart = 2
<camp>.SelLength = 3
```

Exemplu

Să presupunem că avem într-o formă două câmpuri de editare în care utilizatorul poate introduce diferite texte. Forma mai conţine, de asemenea, trei butoane prin intermediul cărora sunt implementate operaţiile de copiere şi de mutare a unui text dintr-un câmp de editare în altul.



*La acționarea butonului **Copiere**, textul selectat din cadrul câmpului curent este copiat într-o variabilă globală. Butonul **Mutare** funcționează asemănător cu **Copiere**, cu deosebirea că textul copiat este șters, după copiere, din locul sursă. În fine, butonul **Inserare** determină introducerea textului copiat sau mutat anterior în variabila globală în cadrul textului.*

*În cadrul formei sunt folosite două variabile globale (de fapt, proprietăți noi ale formei), `camp` și `selectie`. Prima memorează câmpul de editare curent, iar cea de-a doua textul copiat sau mutat. Actualizarea variabilei `camp` se face la părăsirea fiecărui câmp de editare (evenimentul **LostFocus**), iar variabila `selectie` este actualizată la acționarea butoanelor **Copiere** sau **Mutare**.*

*La acționarea butonului **Copiere** este executată următoarea secvență de instrucțiuni:*

```
IF ThisForm.camp=1
   ThisForm.selectie=ThisForm.text1.SelText
ELSE
   ThisForm.selectie=ThisForm.text2.SelText
ENDIF
```

*La acționarea butonului **Mutare** este executată următoarea secvență de instrucțiuni:*

```
IF ThisForm.camp=1
   ThisForm.selectie=ThisForm.text1.SelText
   ThisForm.text1.SelText=""
ELSE
   ThisForm.selectie=ThisForm.text2.SelText
   ThisForm.text2.SelText=""
ENDIF
```

*La acționarea butonului **Inserare** este executată următoarea secvență de instrucțiuni:*

```
IF ThisForm.camp=1
   ThisForm.text1.SelText=ThisForm.selectie
ELSE
   ThisForm.text2.SelText=ThisForm.selectie
ENDIF
```

*Metoda **LostFocus** a fiecărui câmp de editare conține comanda de actualizare a variabilei `camp`:*

`ThisForm.camp=1` *(sau 2, în funcție de câmp)*

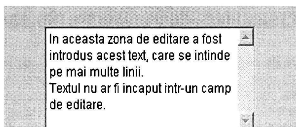
O altă proprietate folosită în operația de selectare a unei porțiuni de text în cadrul unui câmp de editare este **HideSelection**. În mod implicit, ea are valoarea *adevărat*, astfel încât la pierderea controlului de către câmpul de editare, textul selectat nu se mai vede ca fiind în această stare (chiar dacă este). Pentru ca textul să fie afişat ca fiind selectat chiar şi atunci când câmpul de editare nu are ținta intrărilor, trebuie atribuită proprietății **HideSelection** valoarea *fals*.

Proprietatea prezentată mai sus se foloseşte atunci când într-o formă există diferite obiecte de interfață utilizate la manipularea textului selectat într-un câmp de editare. Când unul dintre aceste obiecte primeşte controlul, trebuie ca textul selectat în acel moment să fie vizibil, pentru a şti asupra cărei porțiuni de text se acționează.

În exemplul prezentat mai sus (cel cu copierea şi mutarea), câmpurile de editare au pentru proprietatea **HideSelection** valoarea *fals*.

Zonele de editare a textului (Edit Box)

Spre deosebire de câmpurile de editare, zonele de editare permit introducerea textului pe mai multe rânduri. Prin urmare, aceste obiecte sunt folosite atunci când este necesară preluarea de la utilizator a unor texte de lungimi mai mari, care se întind pe mai multe rânduri.



Introducerea unei zone de editare într-o formă debutează prin acționarea butonului  de pe bara utilitară a obiectelor de interfață. Urmează trasarea cu mouse-ul a zonei din formă pe care o va ocupa noul obiect.

Ca şi la câmpurile de editare, proprietățile **Left**, **Top**, **Width** şi **Height** sunt folosite pentru stabilirea exactă (în pixeli) a zonei de editare din formă. De asemenea, proprietatea **IntegralHeight** poate fi folosită pentru a stabili o înălțime potrivită pentru zona de editare, astfel încât ultimul rând al textului introdus să fie afişat complet pe ecran (în caz contrar, se poate ajunge la situația în care ultimul rând al textului introdus este afişat parțial pe ecran).

Ca şi la câmpurile de editare, prin intermediul proprietății **ControlSource** se specifică variabila sau câmpul de tabelă în care este memorat textul introdus de utilizator.

Derularea pe verticală a conținutului zonei de editare se realizează cu ajutorul barei de derulare din partea dreaptă a obiectului. Această bară poate fi ascunsă (eliminată), caz în care derularea se va face numai cu ajutorul cursorului (deci prin intermediul tastaturii). Proprietatea **ScrollBars** poate lua următoarele valori numerice:

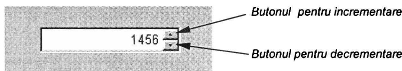
- 0 – care indică lipsa barei de derulare;
- 2 – care indică prezența acesteia.

În general, deplasarea controlului de la un obiect de interfață la altul se face cu ajutorul tastei Tab. În cazul zonelor de editare, pot apărea probleme atunci când se doreşte introducerea în text chiar a caracterului Tab. Pentru a permite acest lucru, se atribuie proprietății **AllowTabs** (se permit caractere Tab) valoarea logică *adevărat*. Ca urmare, acționarea tastei Tab în timp ce zona de editare are ținta intrărilor nu mai determină trecerea la următorul obiect de interfață, ci introducerea caracterului Tab în textul propriu-zis. Pentru ieşirea din zona de editare şi trecerea la următorul obiect de interfață se va folosi în acest caz combinația de taste Ctrl+Tab.

Proprietatea **Text** asociată unei zone de editare este folosită doar pentru citire, ea conținând textul introdus de utilizator, fără caracterele adiționale (precum Rând nou).

Câmpuri de editare cu butoane de incrementare-decrementare (Spinner)

Nişte câmpuri de editare speciale sunt cele cu butoane de incrementare-decrementare. Acestea se utilizează în cazul valorilor numerice, permițând folosirea mouse-ului pentru creşterea, respectiv descreşterea interactivă a valorii din câmpul de editare cu o anumită valoare fixată anterior, la proiectare. Pentru aceasta, câmpul de editare respectiv are în dreapta sa două butoane: unul pentru incrementare şi altul pentru decrementare.



Pe bara utilitară a obiectelor de interfaţă se găseşte butonul , care este folosit pentru definirea acestui tip de obiecte. O dată acţionat acest buton şi trasat în formă câmpul de editare respectiv, se trece la parametrizarea sa.

Ca şi în cazul câmpurilor de editare simple, proprietatea **ControlSource** stabileşte variabila sau câmpul tabelei care memorează valoarea introdusă de utilizator. Alinierea în cadrul câmpului este stabilită de proprietatea **Alignment**, numai că, în acest caz, valoarea implicită este „aliniere la dreapta".

La acţionarea butonului de incrementare sau decrementare, valoarea din câmpul de editare va fi mărită, respectiv micşorată cu o valoare fixă, dată de proprietatea **Increment**, implicit 1.

Pentru câmpurile de editare însoţite de butoane de incrementare-decrementare se pot specifica limitele în care trebuie să se încadreze valoarea introdusă de utilizator. În cazul folosirii tastaturii, limita superioară este dată de **KeyboardHighValue**, iar cea inferioară de **KeyboardLowValue**. Depăşirea acestor valori determină afişarea de către sistem a unui mesaj de eroare.

Dacă pentru creşterea, respectiv descreşterea valorii se foloseşte mouse-ul, cu care se acţionează asupra celor două butoane ale obiectului, limitele sunt date de proprietăţile **SpinnerHighValue** (pentru limita superioară) şi **SpinnerLowValue** (pentru limita inferioară). Depăşirea acestor valori nu conduce la o eroare, dar numărul introdus de utilizator nu mai este mărit, respectiv micşorat.

Depăşirea valorilor impuse de proprietăţile de mai sus pot fi controlate prin intermediul a două evenimente speciale, **RangeHigh** şi **RangeLow**. Ambele evenimente sunt declanşate la pierderea controlului de către câmpul de editare respectiv. Prima dintre metode, **RangeHigh**, poate returna o valoare, cu ajutorul comenzii **RETURN**, valoare semnificând limita superioară care nu trebuie depăşită pentru a se permite trecerea la un nou obiect de interfaţă. În caz contrar, cursorul (ţinta intrărilor) rămâne în câmpul de editare respectiv.

Cu ajutorul metodei **RangeLow**, care de asemenea poate returna o valoare cu comanda **RETURN**, se poate stabili o limită inferioară care trebuie depăşită de valoarea introdusă de utilizator în câmp pentru a se permise cedarea ţintei intrărilor către un nou obiect de interfaţă.

- *adevărat*, pentru ca dimensiunea să fie stabilită automat de sistem, în funcţie de conţinutul obiectului de interfaţă (textul sau imaginea de pe buton);
- *fals*, pentru ca dimensiunea obiectului de interfaţă să fie cea stabilită manual de către proiectant.

În cazul în care butonul are înscris pe el un text, acesta este stabilit prin intermediul proprietăţii **Caption**. Dacă butonul este însă de tip grafic, fişierul imagine (`.BMP`) sau pictograma (`.ICO`) se stabilesc prin proprietatea **Picture**. În acest ultim caz, fişierul trebuie să fi fost creat anterior cu un program de desenare.

În cazul în care pe buton este afişat un text mai mare decât dimensiunea butonului stabilită manual de proiectant, există două posibilităţi:

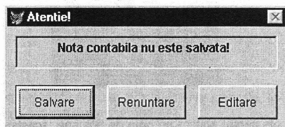
- fie se decupează textul astfel încât pe buton să se afişeze doar cât încape, pentru aceasta atribuindu-se proprietăţii **WordWrap** valoarea *fals*;
- fie se afişează textul respectiv pe mai multe rânduri, pentru aceasta atribuindu-se proprietăţii **WordWrap** valoarea *adevărat*.

În cadrul unei forme există două tipuri de butoane speciale: unul numit „implicit", care este acţionat automat la apăsarea tastei Enter, şi altul numit „de anulare" sau „de renunţare", care este acţionat automat la apăsarea tastei Esc.

Pentru ca un buton să fie implicit, proprietatea **Default** a acestuia trebuie să aibă valoarea *adevărat*. Proprietatea corespunzătoare butonului de renunţare este **Cancel**.

Exemplu

*În următoarea formă, butonul **Salvare** este cel implicit, iar **Renuntare** este butonul de anulare:*



Butoanele pot fi grupate în „grupuri de butoane", ceea ce le conferă o serie de proprietăţi specifice. Un grup de butoane reprezintă un obiect independent compus, cu propriile sale proprietăţi şi metode. De exemplu, există un eveniment **Click** la nivel de grup şi câte un eveniment de acest fel la nivelul fiecărui buton în parte. Dacă se

Prin intermediul metodelor **RangeHigh** şi **RangeLow** se pot modifica limitele între care trebuie să se încadreze valoarea introdusă de utilizator în câmp, aceste limite determinându-se în mod dinamic, la rularea formei.

Exemplu

De exemplu, limitele între care trebuie să se afle retribuţia unui angajat al unei societăţi comerciale pot varia în funcţie de poziţia ocupată în cadrul societăţii, de numărul de zile lucrătoare din lună, de gradul de îndeplinire a planului etc.

Alte două evenimente specifice acestui tip de obiecte de interfaţă sunt **UpClick** şi **DownClick**. Primul dintre ele apare atunci când utilizatorul acţionează cu mouse-ul butonul de incrementare (cel cu săgeata îndreptată în sus), iar cel de-al doilea este asociat butonului de decrementare (cel cu săgeata îndreptată în jos). Cu ajutorul acestor două evenimente se pot realiza diferite operaţii o dată cu incrementarea, respectiv decrementarea valorii din câmp.

Exemplu

De exemplu, dacă valoarea din câmpul de editare reprezintă poziţia unei înregistrări într-o tabelă, la acţionarea unuia dintre cele două butoane se poate realiza mutarea fizică a indicatorului de înregistrări al tabelei pe înregistrarea vizată.

Butoane şi grupuri de butoane (Command şi Command Group)

Unele dintre cele mai folosite obiecte de interfaţă sunt butoanele, care pot fi de diferite dimensiuni şi pot fi plasate fie în forme, fie pe bare utilitare. Ele pot avea înscrise texte sau imagini, care să sugereze operaţia declanşată la acţionarea lor.

Definirea unui buton într-o formă se face prin acţionarea butonului de pe bara utilitară a obiectelor de interfaţă, urmată de trasarea butonului pe formă. O dată definit un buton, el trebuie configurat, adică este necesară stabilirea proprietăţilor şi a reacţiei sale la diferite evenimente.

Dimensiunile şi poziţia butonului pot fi stabilite în mod interactiv de către utilizator (cu mouse-ul), dar şi exact, cu ajutorul proprietăţilor **Left**, **Top**, **Width** şi **Height**. În cazul butoanelor poate fi folosită o proprietate specială pentru determinarea automată a dimensiunilor. Proprietatea se numeşte **AutoSize** (autodimensionare) şi poate lua valori logice:

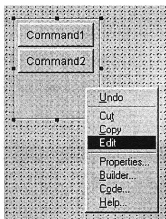
acţionează cu mouse-ul în zona grupului de butoane, dar nu pe un buton anume, va fi declanşată metoda ataşată evenimentului **Click** la nivel de grup.

Definirea unui grup de butoane se realizează prin acţionarea butonului de pe bara utilitară a obiectelor de interfaţă. Urmează trasarea pe formă a zonei ocupate de grupul de butoane. În interiorul acestei zone sunt definite două butoane, dar numărul lor poate fi mărit cu ajutorul proprietăţii **ButtonCount** (contor butoane).

Grupul de butoane reprezintă un obiect complex, de tip container, care conţine mai multe obiecte simple (butoane). Referirea la un obiect de interfaţă din cadrul unui obiect container trebuie precedată de numele obiectului container. Prin urmare, butonul *Command1* din grupul de butoane *CommandGroup1* se desemnează prin construcţia:

```
CommandGroup1.Command1
```

Pentru a opera asupra unui obiect aflat în interiorul unui obiect container, mai întâi trebuie deschis pentru editare obiectul container respectiv. Printr-un clic cu butonul drept al mouse-ului pe obiectul container şi prin alegerea opţiunii **Edit** din meniul afişat pe ecran, se va putea lucra cu obiectele componente.



O dată deschis pentru editare un grup de butoane, acestea pot fi mutate în alte poziţii, obţinând, de exemplu, un grup orizontal de butoane (cel implicit este vertical). Butoanele se pot redimensiona şi se poate schimba textul informativ ataşat fiecăruia în parte. De asemenea, se pot particulariza metodele butoanelor astfel încât acestea să reacţioneze diferit la evenimente.

Principalul eveniment ataşat unui buton este **Click**. Cu ajutorul acestuia se precizează comanda sau grupul de comenzi care să fie executate la acţionarea butonului respectiv. Eveminentul **Click** este declanşat în următoarele situaţii:

- la executarea unui clic cu mouse-ul pe butonul respectiv;
- la acţionarea tastei Enter sau Space atunci când butonul are ţinta intrărilor;
- la acţionarea unei combinaţii speciale de taste ataşate butonului (calea directă de acţionare a butonului respectiv).

Un eveniment asemănător cu **Click** este **DblClick**, ce apare atunci când utilizatorul execută un clic dublu (două apăsări succesive) pe obiectul de interfaţă respectiv.

Clicul dublu pe un anumit obiect de interfaţă este folosit atunci când se doreşte ca o dată cu selectarea unui element să se declanşeze şi o anumită operaţie. De exemplu, în cadrul unei ferestre de alegere a unui fişier de pe disc, se poate alege dintr-o listă fişierul dorit printr-un clic simplu, după care se acţionează un buton din cadrul ferestrei pentru deschiderea fişierului respectiv. Dacă se doreşte deschiderea directă a fişierului respectiv, se poate realiza direct un clic dublu pe el, fără a mai fi nevoie de acţionarea butonului de deschidere.

Acţionarea butonului drept al mouse-ului determină un eveniment **RightClick**, iar a celui din mijloc un eveniment **MiddleClick**. Acest din urmă eveniment apare mai rar, dar evenimentul **RightClick** este des folosit. De obicei, la apăsarea butonului drept al mouse-ului pe un obiect de interfaţă este afişat un meniu din care utilizatorul poate alege operaţia de executat. În general, acest meniu este dependent de context, în sensul că opţiunile sale depind de obiectul de interfaţă pe care s-a executat clicul drept.

Un exemplu a fost prezentat anterior, când realizarea unui clic cu butonul drept pe un obiect container a determinat afişarea unui meniu din care se putea alege opţiunea **Edit** pentru accesul la obiectele de interfaţă componente.

Controlul fin al mouse-ului

Deşi evenimentele prezentate mai sus sunt de cele mai multe ori suficiente, există aplicaţii care necesită un control mai fin al reacţiilor la manipularea mouse-ului. Pentru acestea au fost introduse o serie de evenimente speciale, dintre care aici vor fi prezentate **MouseDown**, **MouseMove** şi **MouseUp**.

Primul dintre evenimente, **MouseDown**, apare atunci când utilizatorul doar apasă butonul mouse-ului, fără a-l elibera, ca în cazul evenimentului **Click**. Eliberarea butonului este însoţită de evenimentul **MouseUp**.

Mişcarea mouse-ului este controlată de evenimentul **MouseMove**. Acest eveniment apare ori de câte ori utilizatorul mută cursorul mouse-ului pe ecran (pe formă sau pe un obiect de interfaţă).

Cele trei metode prezentate sunt apelate de sistem printr-o linie PARAMERTERS de următoarea formă:

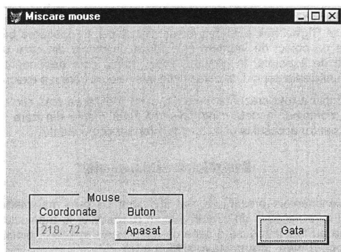
```
LPARAMETERS nButton, nShift, nXCoord, nYCoord
```

în care cei patru parametri au următoarele semnificaţii:

- *nButton* – indică butonul apăsat de utilizator, adică 1 pentru butonul stâng al mouse-ului, 2 pentru butonul drept şi 4 pentru cel din mijloc;
- *nShift* – indică dacă o dată cu butonul mouse-ului au fost apăsate şi tastele adiţionale Shift, Ctrl sau Alt. Modul de calcul al valorii acestui parametru a fost prezentat la câmpurile de editare, la evenimentul **KeyPress**;
- *nXCoord* şi *nYCoord* – reprezintă coordonatele cursorului mouse-ului (pe orizontală şi pe verticală) relativ la formă.

Exemplu

Pentru exemplificarea modului de folosire a metodelor prezentate mai sus, am construit următoarea formă:



*În câmpul de editare sunt afişate permanent coordonatele curente ale cursorului mouse-ului, iar pe **Buton** este afişat textul* Apasat *atunci când butonul mouse-ului este apăsat.*

Pentru a realiza această formă s-au definit mai întâi obiectele de interfaţă. Câmpului de editare în care sunt afişate coordonatele i-a fost atribuită proprietatea şir a formei (o proprietate suplimentară, definită de utilizator special în acest scop): `ThisForm.sir`

La nivel de formă au fost specificate următoarele metode:

***MouseMove**:*

```
LPARAMETERS nButton, nShift, nXCoord, nYCoord
ThisForm.sir=STR(nXCoord,3,0)+", "+STR(nYCoord,3,0)
ThisForm.Text1.Refresh
```

Ori de câte ori se mişcă mouse-ul, în câmpul de editare este afişat un şir de caractere reprezentând coordonatele cursorului. După modificarea proprietăţii `sir`*, aspectul câmpului de editare trebuie reîmprospătat.*

***MouseDown**:*

```
LPARAMETERS nButton, nShift, nXCoord, nYCoord
ThisForm.Command2.Caption="Apasat"
```

*La apăsarea butonului mouse-ului se modifică textul afişat pe butonul **Command2**.*

***MouseUp**:*

```
LPARAMETERS nButton, nShift, nXCoord, nYCoord
ThisForm.Command2.Caption=""
```

*La eliberarea butonului mouse-ului se şterge textul afişat pe butonul **Command2**.*

*Metoda **Click** ataşată butonului **Gata** conţine comanda de închidere a formei, şi anume:*

```
ThisForm.Release
```

Butoane radio sau butoane de selecţie (Radio Button)

Butoanele radio sau butoanele de selecţie reprezintă un tip special de obiecte de interfaţă care permit alegerea unei singure opţiuni din mai multe posibile. Fiecare opţiune are în dreptul ei un cerc, care este gol dacă opţiunea nu este cea aleasă de utilizator şi plin pentru opţiunea curent selectată (a se vedea săgeţile trasate cu linie continuă în figura următoare).

Butoanele de selecţie se găsesc în grupuri. Din cadrul unui grup, un singur buton poate fi selectat la un moment dat. Alegerea unui buton determină deselectarea vechiului buton ales.

Definirea unui grup de butoane radio începe cu acţionarea butonului de pe bara utilitară a obiectelor de interfaţă, după care se trece la trasarea zonei pe care o va ocupa obiectul pe suprafaţa formei.

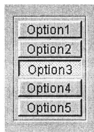
Grupul de butoane radio este un obiect compus (container), deci el conţine mai multe obiecte simple (butoanele radio propriu-zise). La nivel de grup există o serie de proprietăţi şi metode specifice. Ca şi la grupurile de butoane, există proprietatea **AutoSize** care, dacă are valoarea logică *adevărat*, determină calcularea automată a dimensiunii grupului de butoane în funcţie de conţinut (numărul de butoane componente şi textul asociat fiecăruia).

Numărul de butoane conţinute într-un grup este stabilit de proprietatea **ButtonCount**.

Proprietatea **ControlSource**, ca şi la celelalte obiecte de interfaţă, stabileşte variabila sau câmpul tabelei în care este memorat butonul selectat. Variabila trebuie să fie de tip numeric, deoarece în ea se va reţine numărul de ordine al butonului în cadrul grupului de butoane radio.

Dacă se doreşte operarea asupra butoanelor componente, trebuie aleasă opţiunea **Edit** a meniului afişat ca urmare a unui clic executat cu butonul drept pe grupul respectiv. Asupra unui buton de selecţie se pot efectua modificări în ceea ce priveşte aspectul său (dimensiunea, poziţia, textul informativ asociat) sau comportamentul (răspunsul la evenimente precum **Click**, **InteractiveChange** etc.).

Butoanele radio pot apărea pe ecran nu numai sub forma cerculeţelor pline sau goale, ci şi sub formă de butoane care sunt apăsate în cazul unei opţiuni selectate şi neapăsate în cazul celorlalte opţiuni.



Pentru aceasta, trebuie să fie atribuită proprietăţii **Style** a fiecărui buton de selecţie al grupului valoarea *1 – Graphical* (valoarea implicită fiind *0 – Standard*).

Comutatoare (Check Box)

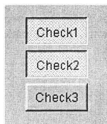
Comutatoarele se folosesc acolo unde trebuie precizată prezenţa sau absenţa unei anumite proprietăţi, respectarea sau nerespectarea unei anumite condiţii şi, în general, acolo unde se poate alege între două stări.

Ele apar pe ecran sub forma unor mici pătrate care au un marcaj în interior atunci când comutatorul este selectat. În figura de pe pagina anterioară comutatoarele sunt puse în evidenţă printr-o săgeată trasată cu linie întreruptă.

Butonul care corespunde comutatoarelor pe bara utilitară a obiectelor de interfaţă este . Acţionarea sa trebuie urmată de indicarea în formă a zonei ocupate de obiectul de interfaţă respectiv.

Comutatoarele au asociate variabile sau câmpuri de tabelă de tip logic, care iau valoarea `.T.` atunci când comutatorul este selectat şi `.F.` în caz contrar. Proprietatea care specifică variabila respectivă este **ControlSource**.

Ca şi în cazul butoanelor de selecţie, există un stil de afişare de tip buton: apăsat când comutatorul este selectat şi neapăsat atunci când comutatorul este neselectat. Proprietatea folosită este **Style** (având ca valori posibile *0 – Standard* şi *1 – Graphical*).



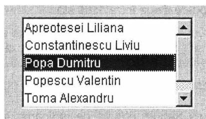
Liste (List Box şi Combo Box)

Def

***Listele** reprezintă un tip special de obiecte, folosite foarte des în interfeţele cu utilizatorul ale sistemelor informatice. Caracteristica principală a listelor este faptul că ele oferă posibilitatea selectării unuia sau mai multor elemente dintr-o mulţime finită, afişată total sau parţial pe ecran.*

Listele pot fi de mai multe feluri. În funcţie de numărul de elemente care sunt afişate în starea dezactivată, listele pot fi:

- *liste simple*, când pe ecran sunt afişate mai multe elemente (nu neapărat toate):

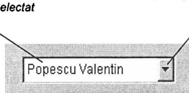


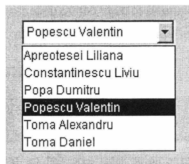
- *liste derulante* sau *expandabile*, caz în care, în stare neactivată, pe ecran este afişat doar elementul curent selectat din listă. Când se doreşte selectarea altui element, se apasă un buton din dreapta listei, pentru ca aceasta să fie desfăşurată, iar utilizatorul să poată face selecţia. Aceste liste pot fi, la rândul lor, de două feluri:

- cu posibilitate de completare manuală a elementului de selectat – acestea permit, pe lângă selectarea din listă a elementului dorit, şi completarea sa manuală, ca în interiorul unui câmp de editare;
- care nu permit specificarea manuală a elementului selectat, ci numai selectarea acestuia din listă.

În acest loc se poate introduce manual elementul de selectat

Aici se apasă pentru a fi deschisă lista





Selectarea unui anumit element din listă se realizează prin executarea unui clic simplu pe acesta sau prin deplasarea cu săgeţile direcţionale ↑ şi ↓ pe elementul dorit şi apăsarea tastei Enter. Dacă lista este derulantă, atunci, pentru a fi selectat un element, ea trebuie mai întâi expandată, lucru care se realizează prin acţionarea cu mouse-ul a butonului asociat (din partea dreaptă), iar cu tastatura prin apăsarea tastei ↓ împreună cu tasta Alt.

Definirea unei liste se face prin acţionarea butonului (**List Box**) de pe bara utilitară a obiectelor de interfaţă (atunci când lista care se creează este una simplă) sau a butonului (**Combo Box**) de pe aceeaşi bară utilitară, în cazul creării unei liste derulante. Urmează trasarea pe formă a zonei care va fi ocupată de obiectul respectiv. Poziţia şi dimensiunile exacte pot fi stabilite cu ajutorul proprietăţilor **Left** (distanţa faţă de latura stângă), **Top** (distanţa faţă de latura superioară), **Width** (lăţimea) şi **Height**

(înălțimea). Proprietatea **IntegralHeight** stabilește o înălțime cuantificată a obiectului, adică o înățime calculată automat astfel încât în listă să se vadă la un moment dat un număr fix de elemente, fără ca ultimul element să fie afișat parțial (pentru că nu încape în întregime).

Culorile sunt controlate de o serie de proprietăți, printre care:

- **ItemBackColor** și **ItemForeColor** stabilesc culorile de fond și de cerneală folosite pentru desenarea articolelor listei, altele decât cel curent selectat;
- **SelectedItemBackColor** și **SelectedItemForeColor** stabilesc culorile de fond și de cerneală folosite pentru desenarea articolului curent selectat din listă;
- **DisabledBackColor** și **DisabledForeColor** se folosesc pentru precizarea culorilor de fond și respectiv de cerneală cu care sunt desenate opțiunile listei, atunci când lista este dezactivată;
- **DisabledItemBackColor** și **DisabledItemForeColor** stabilesc culorile de fond și de cerneală folosite pentru desenarea opțiunilor dezactivate ale listei;
- **BorderColor** precizează culoarea folosită pentru trasarea chenarului listei.

Sursa de date a listei

La construirea unei liste, trebuie stabilită sursa de date a elementelor acesteia, adică de unde se preiau elementele componente. Mai întâi trebuie indicat tipul sursei (proprietatea **RowSourceType**) și apoi sursa efectivă (proprietatea **RowSource**).

Tipul sursei elementelor listei este dat de proprietatea **RowSourceType**, care poate lua următoarele valori:

- *0 – None* (sursă nespecificată) – elementele listei vor fi specificate în mod dinamic, la rularea formei, prin apeluri repetate ale metodei **AddItem** (adăugare articole);
- *1 – Value* (valoare) – elementele listei sunt enumerate manual în proprietatea **RowSource**. Aceasta va conține o listă de elemente separate prin virgulă;
- *2 – Alias* – în acest caz, elementele listei sunt preluate dintr-o tabelă, specificată cu ajutorul proprietății **RowSource**;
- *3 – SQL Statement* (instrucțiune SQL) – elementele listei vor fi cele furnizate de o comandă `SELECT SQL`. Aceasta este o comandă de interogare a unei baze de date, ce poate genera în urma interogării o tabelă permanentă sau una temporară (cursor), care va da elementele listei;
- *4 – Query (.QPR)* (cerere) – o cerere, adică un fișier `.QPR` generat cu ajutorul Constructorului de cereri;

- 5 – *Array* (tablou) – elementele listei sunt preluate dintr-un tablou;
- 6 – *Fields* (câmpuri) – elementele listei reprezintă câmpuri ale unor tabele;
- 7 – *Files* (fişiere) – elementele listei sunt fişiere dintr-un anumit director. Proprietatea **RowSource** specifică macheta pe baza căreia sunt selectate fişierele care vor fi afişate în listă;
- 8 – *Structure* (structură) – lista este alcătuită din câmpurile unei tabele. Numele tabelei este specificat cu ajutorul proprietăţii **RowSource**.

Accesul la elementele listei

Construirea unei liste presupune adăugarea de elemente în listă, unul după altul, în ordinea în care sunt găsite în sursa de date specificată prin intermediul proprietăţilor **RowSource** şi **RowSourceType**.

Obs *În afară de cazul în care elementele listei sunt specificate manual la rularea formei, adăugarea de elemente la listă se face automat, la construirea listei, prin scanarea sursei de date respective.*

Fiecărui element nou adăugat îi sunt atribuite două coduri numerice:

- *poziţia curentă* (**Index**) – reprezentând poziţia elementului în listă la un moment dat. Acest cod se modifică la schimbarea poziţiei elementului în listă, adică la reordonarea elementelor respective, la ştergerea sau adăugarea unor elemente etc.;
- *codul asociat* (**ItemID**) – reprezentând un cod numeric unic asociat fiecărui element în parte la adăugarea sa în listă. Dacă nu se specifică altfel, acest cod reprezintă de fapt poziţia originală a elementului în listă. El nu se modifică la schimbarea poziţiei elementului respectiv în cadrul listei, la reordonare, ştergere sau adăugare de noi elemente etc., ci rămâne acelaşi pe toată perioada cât există elementul respectiv.

Pentru a obţine poziţia curentă a unui element al listei pentru care se cunoaşte codul asociat, se foloseşte metoda **ItemIDToIndex**. Transmiţându-i acestei metode codul elementului respectiv, ea returnează poziţia curentă a acestuia. Metoda menţionată răspunde la întrebarea: „*Care este poziţia curentă a elementului care are codul... ?*"

Funcţia inversă este asigurată de metoda **IndexToItemID**, care realizează trecerea de la poziţia curentă la codul asociat. Această metodă răspunde la întrebarea: „*Care este codul elementului care se află momentan pe poziţia... în listă?*"

Exemplu

*Poziția elementului cu codul 47 în lista **List1** este dată de construcția:*

```
List1.ItemIDToIndex(47)
```

Codul celui de-al treilea element din listă se află cu ajutorul construcției:

```
List1.IndexToItemID(3)
```

Inițial, cele două coduri numerice sunt egale pentru fiecare element al unei liste, dar ele vor deveni diferite la reordonarea listei, la ştergerea sau adăugarea de elemente.

O listă are asociate două proprietăți, numite **List** şi **ListItem**, care permit accesul la elementele componente. Aceste proprietăți sunt de fapt tablouri care au mai multe coloane, în funcție de numărul de coloane din listă.

Fiecare element al listei ocupă o linie a acestor tablouri. Diferența între cele două proprietăți este faptul că în masivul **List** elementele se află în ordinea în care sunt ele afişate pe ecran, adică în poziția lor curentă, pe când în tabloul **ListItem** elementele sunt plasate în ordinea dată de codurile asociate.

Exemplu

*De exemplu, având lista **List1**, construcția:*

```
List1.List[2]
```

identifică al doilea element al listei, în ordinea în care sunt afişate pe ecran elementele respective, iar construcția:

```
List1.ListItem[2]
```

identifică elementul listei care are asociat codul 2.

Ca şi la alte tipuri de obiecte de interfață, proprietatea **ControlSource** indică variabila sau câmpul din tabelă în care se memorează datele referitoare la elementul curent selectat. Dacă tipul acestei variabile este numeric, atunci ea va memora poziția curentă în listă a elementului selectat. Dacă însă tipul variabilei este şir de caractere, atunci în ea va fi depozitat textul asociat elementului selectat din lista respectivă.

Elementul selectat curent în listă este dat şi de proprietățile **ListIndex** şi **ListItemID**. Prima dintre ele returnează poziția curentă a elementului selectat, iar cea de-a doua returnează codul asociat elementului respectiv.

Numărul de elemente din listă este returnat de proprietatea **ListCount**.

Exemplu

*Textul asociat elementului curent selectat din lista **List1** se poate obține prin intermediul construcției:*

```
List1.List[List1.ListIndex]
```

Acelaşi lucru este posibil cu ajutorul următoarei construcții:

```
List1.ListItem[List1.ListItemID]
```

Cele două variante diferă prin modul de acces la elementul selectat. În prima construcție, acest lucru se realizează prin intermediul poziției curente în listă, iar în cel de-al doilea caz se foloseşte codul asociat elementului selectat.

O listă are asociată o proprietate specială, numită **Value** (valoare), care returnează date referitoare la elementul curent selectat, adică textul asociat acestuia sau poziția sa în listă.

Exemplu

*De exemplu, pentru aflarea textului asociat elementului curent selectat din lista **List1**, se poate folosi construcția:*

```
List1.Value
```

Proprietatea **BoundTo** indică tipul de date returnate de proprietatea **Value**:

- dacă **BoundTo** are valoarea `.F.`, atunci **Value** va indica valoarea memorată în variabila specificată în **ControlSource** (textul asociat elementului selectat sau poziția curentă a acestuia în listă);
- dacă **BoundTo** are valoarea `.T.`, atunci **Value** va indica textul asociat elementului curent selectat din listă.

Exemplu

*Să presupunem că lista **List1** are specificată în proprietatea **ControlSource** variabila `v` de tip numeric. Dacă proprietatea **BoundTo** a listei are valoarea `.F.`, proprietatea **Value** va indica poziția în listă a elementului curent selectat. În cazul unei valori `.T.` a proprietății **BoundTo**, **Value** va indica textul opțiunii curent selectate din listă.*

*Între variabila specificată în **ControlSource** şi **Value** pot exista diferențe, determinate, de exemplu, de modificarea prin cod a valorii variabilei respective, fără reîmprospătarea aspectului listei.*

Pentru manipularea mai uşoară a elementelor unei liste (nu prea mari), acest tip de obiecte de interfaţă are asociaţi doi vectori, numiţi **ItemData** şi **ItemIDData**. Cu ajutorul lor se poate asocia fiecărui element al listei câte un cod special, care să fie folosit intern de către proiectant (în funcţie de nevoile sale).

Exemplu

*Am putea de exemplu să avem o listă cu produsele existente într-un magazin, iar în vectorul **ItemData** să memorăm codurile fiecărui produs. La selectarea unui element din listă de către utilizator, cu ajutorul construcţiei:*

```
<listă>.ItemData[<listă>.ListIndex]
```

se obţine direct codul produsului selectat.

Diferenţa între vectorii **ItemData** şi **ItemIDData** este aceea că în primul identificarea elementelor se face prin poziţia în listă, pe când în cel de-al doilea se face după codul special asociat.

Exemplu

Dacă se cunoaşte codul unui produs şi se doreşte poziţia sa în listă, atunci se foloseşte construcţia:

```
<listă>.ItemIDData[<cod>]
```

Adăugarea de elemente noi la o listă este însoţită automat de adăugarea elementelor corespunzătoare la listele **ItemData** şi **ItemIDData**, însă încărcarea vectorilor respectivi cu valori trebuie efectuată manual, prin comenzi de tipul:

```
List1.ItemData[<poziţie>] = <valoare nouă>
List1.ItemIDData[<cod>] = <valoare nouă>
```

Ordinea elementelor din listă

În mod implicit, ordinea elementelor unei liste este cea a adăugării lor la creare. În cazul preluării acestor elemente dintr-o sursă de date, precum un masiv, o tabelă, un cursor etc., ordinea elementelor este dată de sursa de date respectivă.

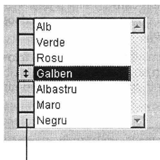
Lista posedă o proprietate specială, numită **Sorted**, care, în starea *adevărat* stabileşte ordinea alfabetică a elementelor sale. Criteriul de ordonare se aplică textului asociat fiecărui element, adică celui furnizat de proprietatea **List**.

Chiar şi în cazul unor liste ordonate, atunci când numărul elementelor este mare, căutarea elementului dorit poate să dureze foarte mult. Pentru a creşte viteza de căutare a unui element într-o listă ordonată se poate folosi o facilitate specială a listelor

căutarea asistată. Aceasta înseamnă că, atunci când lista este activă, acționarea unei taste este interpretată de sistem ca fiind primul caracter al opțiunii căutate și, prin urmare, se realizează automat poziționarea pe prima dintre opțiunile al căror text începe cu caracterul respectiv.

Proprietatea care face ca o listă să dispună de căutarea asistată este **IncrementalSearch**; valoarea *adevărat* a acesteia indică prezența facilității amintite.

Ordinea elementelor într-o listă poate fi stabilită și de utilizator, la rularea formei. Pentru aceasta, trebuie atribuită proprietății **MoverBars** valoarea *adevărat*. În acest caz, în stânga fiecărui element al listei apare un buton, care poate fi folosit pentru mutarea elementului respectiv în mod interactiv în altă poziție a listei.



Aceste butoane se folosesc la deplasarea elementelor listei

Mutarea se poate face astfel:

- cu tastatura – se selectează elementul de mutat și apoi se acționează tasta Ctrl. Cu această tastă apăsată, se deplasează elementul în noua poziție, folosindu-se tastele direcționale ↑ și ↓;
- cu mouse-ul – se trage butonul atașat elementului dorit în noua poziție (se apasă butonul mouse-ului și, ținându-l apăsat, se deplasează cursorul în noua poziție).

Reîmprospătarea listei

Să luăm de exemplu cazul în care datele pentru listă sunt preluate dintr-o tabelă. Ce se întâmplă dacă în tabelă este adăugată o nouă înregistrare? Ar trebui ca și în listă să fie adăugat un nou element. Dar dacă se șterge o înregistrare sau se modifică datele dintr-o înregistrare?

Pentru astfel de situații a fost prevăzută metoda **Requery** (rescanare). La apelarea sa, sursa de date pe baza căreia a fost construită lista este recitită, eventualele modificări fiind transferate în listă.

Metoda **Requery** este analoagă cu metoda **Refresh** aplicată pentru celelalte tipuri de obiecte de interfaţă.

Derularea automată a listei

O listă poate conţine un număr mare de elemente, din care, la un moment dat, pe ecran sunt afişate numai o parte, cele plasate în *zona vizibilă* a listei. Derularea listei în vederea selectării unei opţiuni presupune de fapt schimbarea zonei vizibile a listei cu o zonă ce cuprinde şi opţiunea dorită.

Un obiect de tip listă posedă două proprietăţi cu ajutorul cărora este controlată zona vizibilă. Ele se numesc **TopIndex** şi **TopItemID**. Prima dintre ele indică, prin poziţia curentă în listă, elementul care se află primul în zona vizibilă a listei. Cea de-a doua are aceeaşi utilizare ca şi prima proprietate, cu deosebirea că elementul respectiv este identificat prin codul asociat.

Cele două proprietăţi amintite mai sus sunt folosite atât pentru aflarea zonei vizibile a listei (testarea dacă un anumit element este vizibil pe ecran la un moment dat), cât şi pentru schimbarea acesteia (aducerea unui element al listei în zona vizibilă).

Exemplu

Să luăm, de exemplu, cazul unei forme conţinând o listă şi un buton care realizează selectarea automată a unui anumit element din listă. Acţionarea butonului determină selectarea elementului respectiv prin execuţia unei comenzi de tipul:

```
ThisForm.List1.ListIndex = <poziţie element de selectat>
```

Dar selectarea unui anumit element nu presupune şi afişarea automată a acestuia în zona vizibilă a listei. Pentru aceasta este necesară o comandă suplimentară care să repoziţioneze zona vizibilă a listei astfel încât ea să cuprindă şi elementul selectat. O comandă posibilă ar fi:

```
ThisForm.List1.TopIndex = <poziţie element de selectat>
```

Selectarea multiplă

De obicei, dintr-o listă se selectează un singur element, care este indicat de proprietăţile **ListIndex** sau **ListItemID**. Există însă cazuri când este necesar ca dintr-o listă să fie selectate mai multe elemente. De exemplu, s-ar putea să dorim ca din lista materialelor dintr-un depozit să alegem pentru o comandă mai multe materiale, sau din lista examenelor de susţinut să le alegem pe acelea obligatorii.

Mecanismul implementat în Visual FoxPro permite selectarea mai multor elemente din cadrul unei liste, facilitate disponibilă dacă proprietatea **MultiSelect** a listei are valoarea *adevărat*.

Selectarea mai multor elemente din cadrul unei liste cu această proprietate se poate face astfel:

- cu mouse-ul – ţinând tasta Ctrl apăsată, se execută clic cu mouse-ul pe opţiunile respective;
- cu tastatura – când lista este activată, se deplasează cursorul pe opţiunile de selectat (cu tastele direcţionale ↑ şi ↓), iar pentru selectarea efectivă se apasă combinaţia de taste Ctrl+Space.

Selectarea elementelor din listă are ca scop executarea unei anumite operaţii asupra acestora (copierea în altă listă, ştergerea, efectuarea unor calcule etc.). Pentru a testa prin cod dacă o opţiune a fost sau nu selectată, se folosesc proprietăţile **Selected** şi **SelectedID**.

Aceste două proprietăţi reprezintă de fapt nişte vectori cu atâtea elemente câte are lista respectivă. Dacă elementul este selectat, atunci elementul corespunzător din vectorul **Selected**, respectiv **SelectedID**, va avea valoarea *adevărat*, iar dacă elementul nu este selectat, valoarea din vector va fi *fals*.

Diferenţa dintre cei doi vectori este aceea că elementele din vectorul **Selected** sunt identificate după poziţia curentă în listă, iar cele din vectorul **SelectedID** după codul asociat elementelor din listă.

Exemplu

```
List1.Selected[5]
```

va indica dacă elementul al cincilea din listă (în ordinea de afişare) este selectat, iar construcţia:

```
List1.SelectedID[4]
```

va indica dacă elementul cu codul 4 este selectat.

Exemplu

*Următoarea secvenţă de cod determină afişarea tuturor elementelor selectate de utilizator din lista **List1**:*

```
FOR i=1 TO <formă>.List1.ListCount
  IF <formă>.List1.Selected[i]
    ? List1.List[i]
  ENDIF
ENDFOR
```

Cele două proprietăţi pot fi folosite şi în sens invers, adică pentru selectarea prin cod a elementelor unei liste. Pentru aceasta, se atribuie valoarea *adevărat* elementului corespunzător din vectorul **Selected** sau **SelectedID**.

Exemplu

Comanda:

```
List1.Selected[3]=.T.
```

va selecta elementul al treilea din listă.

Liste multicoloană

Până acum am discutat despre liste alcătuite dintr-o singură coloană, dar există situaţii când în listă trebuie introduse mai multe coloane. Să luăm, de exemplu, o listă cu angajaţii unei unităţi economice, care ar putea conţine în prima coloană marca angajatului, în a doua şi în a treia coloană numele şi prenumele acestuia, iar în cea de-a patra coloană funcţia angajatului.

101	Popa	Alexandru	Director general
102	Toma	Dumitru	Director executiv
103	Popescu	Ioana Maria	Contabil sef
104	Constantinescu	Viorica	Consilier 2
105	Vasiliu	Rodica	Secretara
106	Dimoftache	Constantin	Referent 3
107	Musat	Florica Violeta	Secretara
108	Petrina	Luminita	Inginer sef
109	Elefterescu	Carmen	Referent 2
110	Dumitrescu	Remus	Sef de sectrie

O variantă simplă de a transforma o listă unicoloană într-una multicoloană este aceea de a compune în textul asociat fiecărei opţiuni mai multe date. În exemplul listei cu angajaţii unei societăţi comerciale, se pot preciza următorii parametri:

- **RowSourceType** va avea valoarea *2 – Alias*, indicând astfel că elementele listei vor fi preluate dintr-o tabelă;
- proprietatea **RowSource** va avea valoarea:

```
STR(angajati.marca,8)+'  '+angajati.nume+' '+;
      angajati.prenume+'  '+angajati.functie
```

Deci textul asociat opțiunilor va fi alcătuit prin concatenarea mai multor câmpuri (`marca`, `nume`, `prenume` şi `functie`);

- proprietatea **FontName** va avea valoarea *CourierNew*, adică lista va fi construită cu fontul *CourierNew*. Acest font are toate caracterele de aceeaşi lăţime, ceea ce este necesar pentru ca toate coloanele (numele, prenumele şi funcţiile) să aibă datele plasate unele sub altele.

101	Popa	Alexandru	Director general
102	Toma	Dumitru	Director executiv
103	Popescu	Ioana Maria	Contabil sef
104	Constantinescu	Viorica	Consilier 2
105	Vasiliu	Rodica	Secretara
106	Dimoftache	Constantin	Referent 3
107	Musat	Florica Violeta	Secretara
108	Petrina	Luminita	Inginer sef
109	Elefterescu	Carmen	Referent 2
110	Dumitrescu	Remus	Sef de sectrie

Aceasta este o listă unicoloană, dar datorită alinierii stricte apare ca multicoloană

Tehnica prezentată mai sus este simplă. Dacă se doreşte însă ca lista să aibă efectiv mai multe coloane, trebuie modificată proprietatea **ColumnCount**. Aceasta poate lua o valoare numerică ce indică numărul de coloane din listă. Urmează specificarea în proprietatea **ColumnWidths** a unei liste de numere, separate prin virgulă, semnificând lăţimea coloanelor listei.

Liniile separatoare dintre coloane sunt afişate numai dacă proprietatea **ColumnLines** are valoarea *adevărat*, iar în caz contrar nu sunt afişate (chiar dacă împărţirea pe coloane se păstrează).

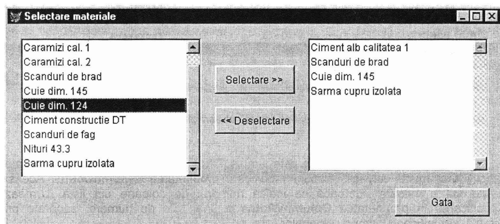
În cazul unor liste multicoloană, proprietatea **Value** returnează în mod implicit textul elementului selectat din prima coloană a listei. Dacă se doreşte ca în proprietatea **Value** să fie preluat textul altei coloane a listei, este necesar ca această coloană să fie specificată în proprietatea **BoundColumn**.

În cazul în care în **Value** este preluată o altă coloană decât prima, dar se doreşte totuşi textul din prima coloană, se va folosi proprietatea **DisplayValue**. Aceasta returnează textul asociat elementului selectat din prima coloană a listei.

Liste construite dinamic

Un tip special de liste sunt cele pentru care elementele nu se specifică la proiectare, ci sunt adăugate în mod dinamic la rularea formei. Să luăm, de exemplu, o formă proiectată pentru alegerea unor materiale dintr-un depozit, materiale care urmează a fi trecute pe o factură.

Un mod de construire a unei astfel de forme ar putea fi prin intermediul a două liste, una conţinând materialele existente în depozit, iar alta materialele selectate de utilizator pentru a face obiectul facturii respective. Cea de-a două listă nu conţine iniţial nici un element, ci se completează în mod dinamic (la rularea formei), pe măsură ce utilizatorul selectează elemente din prima listă.



Listele completate la rularea formei se caracterizează prin valoarea *0 – None* a proprietăţii **RowSourceType** (acesta este cazul celei de-a doua liste a formei din figura de mai sus).

Pentru adăugarea unui element în listă se foloseşte una dintre metodele **AddItem** sau **AddListItem**. Prima dintre ele se apelează cu următorii parametri:

```
<obiect>.AddItem(<text>, [<poziţie>], [<coloană>])
```

şi reprezintă:

- textul care va fi asociat noului element şi care va fi afişat în listă pe poziţia elementului respectiv;

- poziția în listă unde va fi introdus noul element. Dacă nu se specifică nimic, elementul va fi adăugat la sfârşitul listei;
- coloana pentru care se specifică textul respectiv (evident, numai în cazul listelor multicoloană).

O metodă echivalentă cu `AddItem` este `AddListItem`, aceasta din urmă folosindu-se atunci când elementul de adăugat este identificat prin codul asociat şi nu prin poziția lui în listă. `AddListItem` se apelează cu următoarea structură:

```
<obiect>.AddListItem(<text>,[<cod asociat>],[<coloană>])
```

Pentru a afla poziția în care a fost introdus cel mai nou element al listei se foloseşte proprietatea `NewIndex`. Proprietatea analoagă pentru aflarea codului asociat ultimului element adăugat este `NewItemID`. Aceste două proprietăți sunt utile mai ales atunci când se adaugă elemente la o listă sortată (proprietatea `Sorted` având valoarea *adevărat*), caz în care adăugarea elementelor nu se mai face la sfârşitul listei, ci în poziția în care se găsesc acestea prin ordonare.

Exemplu

*Următorul text specifică poziția în listă şi codul asociat ultimului element adăugat la lista **List1**:*

```
"Ultimul element a fost adaugat in pozitia "+;
  STR(List1.NewIndex,4,0)+" si are codul "+List1.NewItemID
```

*Acest text se poate introduce, de exemplu, în proprietatea **Caption** a unui text informativ al formei respective.*

Înlăturarea unui element din listă se realizează cu ajutorul metodelor `RemoveItem` sau `RemoveListItem`. Metodele se apelează printr-o sintaxă de forma:

```
RemoveItem(<poziţie>)
RemoveListItem(<cod asociat>)
```

Înlăturarea tuturor elementelor din listă şi deci golirea completă a acesteia se realizează cu ajutorul metodei **Clear**.

Exemplu

Vom vedea, pe scurt, modul în care a fost construită forma pentru selectarea materialelor (prezentată anterior).

*Lista din stânga formei este una obişnuită, cu elementele preluate dintr-o tabelă de materiale. Lista din dreapta este însă construită la rulare, prin urmare proprietatea sa **RecordSourceType** are valoarea* 0 – None.

*La acționarea butonului **Selectare** (prin metoda **Click** a acestuia), este executată comanda:*

```
ThisForm.List2.AddItem(ThisForm.List1.Value)
```

Comanda determină adăugarea la lista din dreapta a unui nou element şi asocierea la acesta a textului elementului selectat din lista din stânga.

*La acționarea butonului **Deselectare** este executată comanda:*

```
ThisForm.List2.RemoveItem(ThisForm.List2.ListIndex)
```

*care are ca efect ştergerea elementului curent selectat din lista **List2**.*

*În formă ar mai fi putut fi inclus un buton prin intermediul căruia să se golească automat lista din dreapta. Butonul ar fi avut în metoda **Click** asociată comanda de golire a listei, adică un apel la metoda **Clear** a listei respective.*

Lista cu elementele specificate manual

Acest tip de listă se foloseşte atunci când opțiunile componente sunt cunoscute apriori (la proiectare) şi sunt în număr mic.

Proprietatea care stabileşte că opțiunile vor fi specificate manual este **RecordSourceType**, care trebuie să aibă valoarea *1 – Value*. În acest caz, pentru specificarea elementelor componente ale listei se foloseşte proprietatea **RecordSource**, care va conține o listă a elementelor respective, separate prin virgulă.

Lista cu date preluate din tablouri

O altă sursă de date foarte des folosită este tabloul. Un tablou poate furniza elementele unei liste. Tabloul poate fi unidimensional (vector), caz în care lista este şi ea unicoloană, sau bidimensional (matrice), caz în care pe baza lui se poate construi o listă multicoloană: fiecare coloană a matricei va furniza date pentru o coloană a listei; fiecare linie a tabloului va genera un element în listă.

Nu este obligatoriu ca dintr-un tablou să fie preluate în listă toate liniile acestuia. Dacă se doreşte ca sursa de date pentru listă să fie doar un subinterval de linii ale tabloului, se folosesc două proprietăți speciale, şi anume **FirstElement** (primul element) şi **NumberOfElements** (numărul de elemente). Prima dintre proprietăți indică linia de la care se începe preluarea elementelor tabloului în listă (inclusiv), iar cea de-a doua indică numărul de elemente preluate din tablou.

Listele cu elementele preluate din tablouri sunt des folosite datorită facilităților disponibile la prelucrarea tablourilor. De exemplu, dintr-un tablou se poate şterge foarte

uşor o linie sau mai multe, se poate adăuga o linie sau o coloană, modifica valoarea unui element etc.

Liste derulante sau expandabile

Diferenţa esenţială dintre o listă simplă şi una derulantă este faptul că, în cel de-al doilea caz, în starea neactivată, este afişat numai elementul curent selectat din listă. Când se doreşte selectarea unui alt element, lista se desfăşoară (pentru ca zona vizibilă a listei să devină mai mare).

Listele derulante se folosesc acolo unde spaţiul disponibil pentru obiectul de interfaţă respectiv este restrâns, deoarece aceste obiecte afişează, în starea dezactivată, doar un singur element al listei. Chiar dacă pentru selecţia altui element lista trebuie extinsă, acest lucru se face doar pe perioada operaţiei respective.

Popescu Valentin

Listele derulante sunt folosite frecvent pe barele utilitare, deoarece înălţimea mică a acestora permite doar astfel de liste.

Listele derulante sunt şi ele de două tipuri:

- cele care permit introducerea manuală a elementului ce se doreşte a fi selectat. În acest caz, obiectul de interfaţă reprezintă de fapt o combinaţie între listă derulantă şi câmp de editare. Selectarea din listă a unui element conduce la completarea zonei de editare a obiectului cu textul asociat elementului respectiv;
- cele care nu permit completarea manuală a elementului de selectat. În acest caz, schimbarea elementului activ se face doar prin selectarea din listă.

Selectarea între cele două tipuri de liste se face cu ajutorul proprietăţii **Style**. Aceasta poate lua două valori:

- *0 – Dropdown Combo* – pentru listele ce permit completarea manuală;
- *2 – Dropdown List* – în cazul listelor ce nu permit introducerea manuală a elementului de selectat.

Celelalte proprietăţi şi metode prezentate la listele simple sunt valabile şi în cazul listelor derulante.

Grile (Grid)

Def *Grilele reprezintă obiecte de interfaţă complexe, care au în componenţă mai multe obiecte de interfaţă, de obicei câmpuri de editare, plasate într-o matrice. Grilele sunt folosite acolo unde se doreşte editarea conţinutului mai multor înregistrări (eventual în număr variabil), fiecare conţinând mai multe câmpuri.*

Nota model

Numar 1

Descriere

Explicatii	Debit	Credit	Formula	...
				...

N.conturi | Salvare | Tiparire | Stergere | Terminare

Aceasta este o grilă

Să luăm exemplul unui program de introducere de date cu ajutorul căruia s editează conţinutul unei tabele (se adaugă, se şterg sau se modifică înregistrări) Varianta clasică a programului era reprezentată de fereastra **Browse**, care era de fapt o fereastră în care pe coloane aveam câmpurile tabelei, iar pe linii înregistrările acesteia. La intersecţia unei linii cu o coloană se găsea un câmp de editare, în care se introducea sau se edita conţinutul tabelei.

Grila reprezintă o variantă îmbunătăţită a ferestrei **Browse**, în sensul că ea est tratată ca un obiect, cu proprietăţi şi metode specifice. Rezultatul este un control ma

bun asupra acesteia, o mai bună parametrizare a sa în vederea obţinerii unor comportamente speciale. Un exemplu în acest sens este posibilitatea de introducere în grilă şi a altor tipuri de obiecte de interfaţă, nu numai a câmpurilor de editare. De exemplu, putem avea într-o tabelă un câmp de tip logic, pentru care este mai potrivit un comutator decât un câmp de editare.

Aici se află un comutator

Termene | Plati | Contari

Fixat	Data	Valoare	%
☑ fixat la data	11/12/1998	1456800.00	0.0000

Total 0.00

Un alt exemplu ar fi acela al unui câmp în care avem codificate numeric culori. În cazul ferestrei **Browse**, pe poziţia câmpului respectiv se află un câmp de editare în care utilizatorul trebuie să introducă, cifră cu cifră, codul culorii dorite. O variantă mai bună ar fi ca în locul câmpului de editare respectiv să se găsească o listă derulantă, din care utilizatorul să-şi aleagă culoarea dorită (care acum este scrisă explicit).

Definirea unei grile se realizează prin acţionarea butonului de pe bara utilitară a obiectelor de interfaţă, urmată de trasarea zonei din formă care va fi ocupată de grilă.

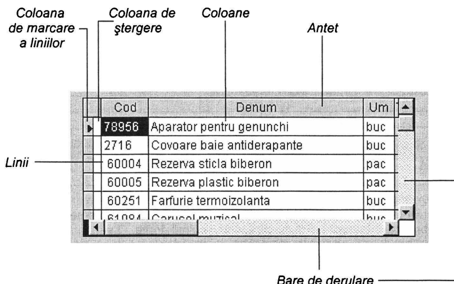
Componentele unei grile

Componentele unei grile sunt:

- *liniile*, *coloanele* şi *celulele* acesteia – o grilă reprezintă de fapt o matrice în care sunt plasate diferite obiecte de interfaţă. La intersecţia unei linii cu o coloană se află o celulă;

- *barele de derulare* – folosite pentru derularea zonei vizibile a grilei, astfel înc liniile sau coloanele dorite să fie accesibile atunci când dimensiunea matric depăşeşte dimensiunea zonei alocate obiectului de interfaţă în formă;
- *antetul* grilei – folosit pentru afişarea capului de tabel, adică a semnificaţ coloanelor componente ale grilei;
- *coloana de marcare a liniilor*;
- *coloana de ştergere a liniilor*.

Componentele unei grile sunt prezentate în următoarea figură:



Acestea pot să nu fie toate prezente la o anumită grilă, în funcţie de valori anumitor proprietăţi.

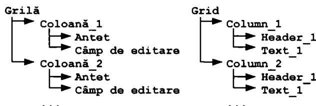
Ca şi fereastra **Browse**, o grilă se poate împărţi în două secţiuni independente. acest caz apar câteva elemente suplimentare, şi anume:

- *secţiunile grilei* – cele două părţi ale grilei, care funcţionează fiecare ca subgrilă independentă;
- *bara de separare a celor două secţiuni* ale grilei.

Organizarea internă a grilei. Referirea la elementele grilei

O grilă este un obiect complex, care poate conține, la rândul lui, alte obiecte. Prin urmare, parametrizarea unei grile (aspect şi comportament) presupune parametrizarea atât a grilei în ansamblul său, cât şi a elementelor componente.

La primul nivel, o grilă conține mai multe coloane. Fiecare dintre coloane conține, la rândul ei, două elemente: antetul coloanei şi obiectul de interfață. Acesta din urmă este de cele mai multe ori un câmp de editare, dar poate fi şi un comutator, o listă derulantă etc. Obiectul respectiv este conținut în obiectul coloană. Prin urmare, avem următoarea ierarhie:



Fiecare dintre elementele prezentate are proprietăți şi metode asociate, care trebuie parametrizate în funcție de aspectul şi comportamentul dorit. De exemplu, pentru ca titlul unei coloane să fie plasat în centru, trebuie ca proprietatea **Alignment** a antetului coloanei respective, **Header**, să aibă valoarea *2 – Middle Center* (mijloc, centru).

Referirea la o proprietate a unui obiect trebuie precedată de obiectele în care acesta este încorporat.

Exemplu

*De exemplu, proprietatea **ControlSource** a câmpului de editare **Text1**, plasat în coloana a doua a grilei **Grid1**, se face prin construcția:*

```
Grid1.Column2.Text1.ControlSource
```

O metodă specifică obiectelor de interfață compuse este **SetAll**. Cu ajutorul ei se pot modifica valorile aceleiaşi proprietăți pentru toate obiectele încorporate în obiectul compus respectiv. Sintaxa metodei este următoarea:

```
<obiect compus>.SetAll(<proprietate>, <valoare>)
```

`<proprietate>` reprezintă un şir de caractere indicând proprietatea care se modifică, iar `<valoare>` este noua valoare care se atribuie tuturor proprietăților `<proprietate>` ale componentelor lui `<obiect compus>`.

Exemplu

De exemplu, pentru schimbarea fontului tuturor obiectelor grilei se poate folosi comanda:

```
Grid1.SetAll("FontName","Arial")
```

Pe lângă metoda prezentată mai sus, referirea la coloanele unei grile (la proprietăţile şi metodele acesteia) se poate face şi cu ajutorul unui vector special, numit **Columns**, asociat oricărui obiect de tip grilă. Referirea se face astfel:

```
<grilă>.Columns[<număr coloană>].<proprietate sau metodă>
```

Exemplu

```
Grid1.Columns[3].DataSource="angajati.nume"
```

Construcţia de mai sus determină atribuirea valorii `angajati.nume` *pentru proprietatea* ***DataSource*** *a coloanei a treia din grila* ***Grid1****.*

Numărul de coloane ale grilei este dat de proprietatea **ColumnCount**. În cazul în care sursa de date conţine mai multe câmpuri, iar proprietatea **ColumnCount** are valoarea –1, numărul de coloane ale grilei va fi ajustat, astfel încât să existe câte o coloană pentru fiecare câmp.

Celula activă este desemnată prin intermediul proprietăţilor **ActiveRow** şi **ActiveColumn**. Prima dintre ele indică linia pe care se află celula activă, iar cea de-a doua coloana care conţine această celulă.

Aspectul unei grile

O dată definită o grilă, acesteia i se alocă în formă o zonă delimitată exact de proprietăţile **Left** (distanţa faţă de latura stângă a formei), **Top** (distanţa faţă de latura superioară a formei), **Width** (lăţimea) şi **Height** (înălţimea).

Înălţimea antetului este dată de proprietatea **HeaderHeight**. Modificarea interactivă a acestei dimensiuni de către utilizator la rularea formei este posibilă numai dacă proprietatea **AllowHeaderSizing** are valoarea logică *adevărat*.

Înălţimea liniilor grilei este stabilită prin intermediul proprietăţii **RowHeight**. Ca şi în cazul antetului, există o proprietate, numită **AllowRowSizing**, care, atunci când are valoarea *adevărat*, permite utilizatorului modificarea interactivă a acestei dimensiuni (la rularea formei).

Liniile şi coloanele grilei sunt separate printr-o linie simplă, care este vizibilă numai atunci când proprietatea **GridLines** are valoarea *adevărat*. Grosimea acestei linii este stabilită de **GridLinesWidth**, iar pentru culoarea liniei se foloseşte proprietatea **GridLineColor**.

Într-o grilă, se pot introduce date la un moment dat într-o singură celulă, numită celula activă. Aceasta este pusă în evidenţă printr-un chenar diferit de al celorlalte. Când se modifică celula activă, textul noii celule poate fi selectat automat, pentru a fi înlocuit direct cu un nou conţinut. Pentru aceasta, se atribuie proprietăţii **Highlight** valoarea *adevărat* (care este şi varianta implicită).

Această proprietate este analoagă proprietăţii **SelectOnEntry** de la câmpurile de editare.

Sursa de date

Datele din grilă sunt preluate şi, în final, memorate în sursa de date asociată. Tipul sursei este dat de **RecordSourceType**, iar denumirea exactă a acesteia se găseşte în proprietatea **RecordSource**.

Ca sursă de date pentru o grilă putem avea o tabelă permanentă sau temporară. Aceasta este fie deja construită, fie se construieşte la iniţializarea grilei, pe baza unei cereri, a unei instrucţiuni SQL etc., în funcţie de valoarea asociată proprietăţii **RecordSourceType**:

- *0 – Table* şi *1 – Alias* – sursa de date este o tabelă deja construită, care va fi deschisă automat la iniţializarea grilei. În acest caz, înregistrările tabelei vor fi linii în grilă, iar câmpurile tabelei vor fi coloane;
- *2 – Prompt* – sursa de date se specifică manual la rularea formei. Utilizatorul este chestionat la rulare despre tabela care va furniza datele pentru grilă;
- *3 – Query (.QPR)* – sursa de date este o cerere (fişier cu extensia .QPR) generată cu ajutorul Constructorului de cereri. Fişierul .QPR este indicat de proprietatea **RecordSource**;
- *4 – SQL Statement* – o instrucţiune SQL de interogare a unei baze de date va furniza rezultatele în grilă; instrucţiunea este specificată prin intermediul proprietăţii **RecordSource**.

O grilă are mai multe coloane, fiecare dintre ele conţinând un obiect de interfaţă (de cele mai multe ori, un câmp de editare). Fiecare obiect de interfaţă are propria sa sursă de date, controlată, în general, de proprietatea **ControlSource**. Sursa poate fi un câmp al unei tabele, o variabilă, un element de tablou etc.

De exemplu, dacă o grilă preia datele dintr-o tabelă, atunci pentru fiecare dintre obiectele de interfaţă conţinute în coloanele grilei trebuie specificat câmpul tabelei care va fi editat în coloana respectivă (în proprietatea **ControlSource**).

Derularea grilei

Numărul de linii ale unei grile poate depăşi dimensiunea verticală a acesteia şi de asemenea, numărul de coloane ale grilei poate depăşi dimensiunea sa orizontală. Este motivul pentru care au fost prevăzute barele de derulare, cu ajutorul cărora se poate deplasa conţinutul grilei în sus şi în jos, la dreapta şi la stânga, pentru a se asigura accesul la toate liniile şi coloanele grilei.

Prezenţa barelor de derulare este stabilită de proprietatea **ScrollBars**, care poate avea următoarele valori:

- *0 – None* – nu va fi afişată nici o bară de derulare, iar operaţia va fi realizată doar prin intermediul tastaturii (prin încercarea de deplasare a cursorului dincolo de capetele vizibile ale grilei);
- *1 – Horizontal* – va fi afişată doar bara de derulare orizontală, care permite accesul cu mouse-ul la toate coloanele grilei;
- *2 – Vertical* – în acest caz, va fi prezentă doar bara de derulare verticală, care va oferi utilizatorului posibilitatea de acces cu mouse-ul la toate liniile grilei;
- *3 – Both* – în acest caz, vor fi prezente ambele bare de derulare.

Pentru realizarea de diferite operaţii la derularea grilei se foloseşte metoda **Scrolled**, asociată evenimentului cu acelaşi nume. Ori de câte ori utilizatorul derulează conţinutul grilei (cu barele de derulare sau cu tastatura), sistemul generează un eveniment **Scrolled**.

Metoda **Scrolled** este apelată de sistem cu un parametru care indică modul în care a fost realizată derularea. Prin urmare, prima linie a metodei este una de tip `PARAMETERS`:

```
PARAMETERS <mod de derulare>
```

Parametrul transmis funcţiei are următoarea semnificaţie:

- *0* – tasta ↑;
- *1* – tasta ↓;
- *2* – acţionarea cu mouse-ul a barei de derulare verticală în zona de s… (derulare în jos a conţinutului grilei);
- *3* – acţionarea cu mouse-ul a barei de derulare verticală în zona de … (derulare în sus a conţinutului grilei);
- *4* – tasta ←;
- *5* – tasta →;

- *6* – acționarea cu mouse-ul a barei de derulare orizontală în zona din stânga (derulare spre dreapta a conținutului grilei);
- *7* – acționarea cu mouse-ul a barei de derulare orizontală în zona din dreapta (derulare spre stânga a conținutului grilei).

O grilă are asociată o metodă specială, numită **DoScroll** (realizează derulare), care realizează derularea grilei prin cod (prin intermediul comenzilor introduse în diferite metode). Metoda se apelează cu un parametru care indică tipul de derulare ce va fi realizat:

```
<Grilă>.DoScroll(<tip derulare>)
```

- *0* – derulare o linie în sus;
- *1* – derulare o linie în jos;
- *2* – derulare o pagină în sus;
- *3* – derulare o pagină în jos;
- *4* – derulare o coloană în stânga;
- *5* – derulare o coloană în dreapta;
- *6* – derulare o pagină spre stânga;
- *7* – derulare o pagină spre dreapta.

Exemplu

De exemplu, instrucțiunea

```
<grilă>.DoScroll(1)
```

va derula grila cu o linie în jos.

Pentru aflarea zonei vizibile din cadrul unei grile se folosesc proprietățile **RelativeRow** şi **RelativeColumn**. Prima dintre ele indică numărul liniei care conține celula activă relativ la prima linie afişată în zona vizibilă a grilei, iar cea de-a doua precizează coloana activă, calculul efectuându-se relativ la coloana cea mai din stânga a zonei vizibile a grilei.

Exemplu

*De exemplu, dacă **RelativeRow** are valoarea 1, înseamnă că celula activă se află pe prima linie vizibilă a grilei, iar dacă **RelativeColumn** are valoarea 3, celula activă se află în a treia coloană care se vede în grilă (chiar dacă în stânga coloanelor vizibile se află şi alte coloane).*

Combinând proprietățile **ActiveRow**, **ActiveColumn** şi **RelativeRow** şi **RelativeColumn** se pot obține diferite informații utile.

Exemplu

```
ActiveRow-RelativeRow+1
```

indică numărul primei linii vizibile a grilei, chiar dacă aceasta nu conţine celula activă. La fel:

```
ActiveColumn-RelativeColumn+1
```

indică numărul primei coloane vizibile a grilei.

RelativeRow şi **RelativeColumn** nu pot fi folosite pentru schimbarea celul active sau pentru modificarea zonei vizibile a grilei (derulare).

Grilă cu date preluate dintr-o tabelă

Folosirea unei tabele ca sursă de date pentru o grilă este, poate, cel mai de întâlnit caz în practică. Stabilirea acestei surse de date se face atribuind proprietă **RecordSourceType** valoarea *0 – Table*. Proprietatea **RecordSource** se foloseşte î acest caz pentru specificarea tabelei respective.

Exemplu

Marca	Nume	Prenume	Functia
101	Popa	Alexandru	Director g
102	Toma	Dumitru	Director e
103	Popescu	Ioana Maria	Contabil
104	Constantinescu	Viorica	Consilier
105	Vasiliu	Rodica	Secretara
106	Dimoftache	Constantin	Referent
107	Musat	Florica Violeta	Secretara

De exemplu, în grila de mai sus sunt preluate date din tabela angajaţilor `ANGAJATI.DBF`. *Pentru a realiza acest lucru, mai întâi a fost definită grila în formă, apoi s-a stabilit valoarea* 0–Table *pentru proprietatea* ***RowSourceType****. Proprietăţii* ***RowSource*** *i-a fost atribuită valoarea* `angajati`.

Prin intermediul grilei se pot modifica (edita) datele din tabela de bază. În plus, se pot şterge sau adăuga înregistrări în această tabelă.

Adăugarea de înregistrări la tabelă prin intermediul grilei este posibilă numai atunci când proprietatea **AllowAddNew** (se permite adăugarea) are valoarea logică *adevărat*. În acest caz, pentru adăugarea unei noi înregistrări, se poziţionează cursorul în ultima linie a grilei (şi deci în ultima înregistrare a tabelei) şi apoi se apasă tasta ↓. Deoarece nu mai există linii pe care să se deplaseze cursorul, în grilă va fi adăugată o nouă linie şi, o dată cu aceasta, o înregistrare nouă în tabelă.

Ştergerea interactivă a unei înregistrări dintr-o tabelă prin intermediul grilei asociate este posibilă atunci când grila conţine în partea sa stângă o coloană specială, numită „coloană de ştergere". Prezenţa acestei coloane este stabilită de proprietatea **DeleteColumn**, care trebuie să aibă valoarea *adevărat* pentru ca grila să includă această coloană. În acest caz, ştergerea unei înregistrări de către utilizator se face printr-un clic simplu cu mouse-ul pe butonul corespunzător înregistrării respective din coloana de ştergere. În urma acestei acţiuni va fi activat marcatorul de ştergere logică al înregistrării respective în tabela sursă.

Ştergerea unei înregistrări dintr-o grilă este însoţită în Visual FoxPro de un eveniment special, numit **Deleted**, care are asociată o metodă a grilei. În cadrul acestei metode pot fi incluse diverse comenzi care să fie executate la ştergerea unei linii a grilei (a unei înregistrări a tabelei). Metoda este apelată de sistem cu un parametru, care indică numărul înregistrării marcate pentru ştergere.

Exemplu

*De exemplu, dacă se doreşte ca la ştergerea unui director din tabela angajaţilor să fie afişat un mesaj de avertizare, se va folosi metoda **Deleted**. În cadrul ei se va testa înregistrarea respectivă, pentru a vedea dacă este vorba de un director şi, în caz afirmativ, se va cere afişarea mesajului.*

O altă coloană asemănătoare coloanei de ştergere este cea pentru selecţia înregistrărilor. Pentru a fi afişată această coloană, proprietatea **RecordMark** trebuie să aibă valoarea *adevărat*.

Grilă construită în mod dinamic, la rulare

O grilă pentru care nu se specifică nici o coloană (**ColumnCount** este 0) se presupune că va fi completată la rularea formei. Prin urmare, la o astfel de grilă trebuie adăugate coloane prin cod. Liniile vor fi preluate tot dintr-o tabelă, care este specificată prin intermediul proprietăţii **RecordSource**.

Adăugarea unei coloane la o grilă se realizează cu ajutorul metodei **AddColumn**. Aceasta are următoarea sintaxă:

```
AddColumn(<poziţie coloană>)
```

Metoda se apelează cu un parametru care indică poziţia noii coloane în cadrul gril (celelalte coloane vor fi deplasate corespunzător, pentru a face loc celei noi).

Ştergerea uneia sau a mai multor coloane se face prin micşorarea numărului d proprietatea **ColumnCount**. Ca urmare, vor fi eliminate ultimele coloane ale grilei (î funcţie de cât se micşorează **ColumnCount**). Dacă se doreşte ştergerea unei coloan din interiorul grilei, aceasta trebuie mai întâi deplasată la sfârşitul grilei şi apoi ştear ca mai sus. Pentru deplasarea unei coloane a grilei pe ultima poziţie se foloseş proprietatea **ColumnOrder** a coloanei respective, care indică poziţia sa în grilă.

Exemplu

De exemplu, comanda

```
<grilă>.Column_x.ColumnOrder=<grilă>.ColumnCount
```

*realizează trecerea coloanei **Column_x** pe ultima poziţie.*

Adăugarea prin cod a unui nou obiect la un obiect compus se realizează c metoda **AddObject**. Acesta este şi cazul adăugării unui obiect de interfaţă la o coloan a unei grile, deoarece o coloană este un obiect compus, care va conţine obiectul d interfaţă respectiv.

Sintaxa metodei **AddObject** este următoarea:

```
AddObject(<nume>,<clasă>,<clasă OLE>,<param.1>,<param.2>,...)
```

În această sintaxă, `<nume>` reprezintă numele obiectului ce se adaugă, i `<clasă>` reprezintă clasa (a se vedea capitolul referitor la programarea orientată spr obiecte) pe baza căreia se creează obiectul respectiv.

Parametrii `<param.1>`, `<param.2>`... sunt, de fapt, parametri transmişi metod **Init** a obiectului de interfaţă nou creat (se ştie că la crearea unui obiect este apela automat procedura sa **Init**).

Exemplu

De exemplu, comanda

```
<grilă>.<coloană>.AddObject("Ob_1","ComboBox")
```

*determină adăugarea la coloana `<coloană>` a grilei `<grilă>` a unui obiect de interfaţă de tip listă derulantă (clasa este **ComboBox**), care se va numi `Ob_1`.*

Ştergerea unui obiect dintr-un obiect compus se face cu ajutorul metodei **RemoveObject** a obiectului compus respectiv. Metoda este apelată cu un parametru, numele obiectului care va fi şters.

Exemplu

Ştergerea unui obiect de interfaţă al unei coloane se poate face printr-o construcţie de forma:

```
<grilă>.<coloană>.RemoveObject(<nume obiect de şters>)
```

Parametrizarea coloanelor grilei

O grilă poate conţine mai multe coloane, fiecare cu caracteristici proprii. Prin urmare, atunci când se doreşte ca o coloană să aibă anumite proprietăţi, diferite de cele implicite, trebuie modificate proprietăţile şi metodele sale (acest lucru este posibil, deoarece o coloană este un obiect inclus în obiectul compus de tip grilă). La rândul ei, o coloană este obiect compus, ea conţinând un obiect de tip antet şi un obiect de interfaţă. La fel, dacă se doreşte o parametrizare detaliată a acestor elemente, trebuie modificate proprietăţile şi metodele corespunzătoare.

Obs

*Pentru a avea acces la coloanele unei grile, trebuie ca numărul de coloane dat de proprietatea **ColumnCount** să fie diferit de –1, valoare care stabileşte calcularea automată a numărului de coloane în funcţie de sursa de date.*

*În acest caz, în fereastra proprietăţilor din Constructorul de forme, grila va avea ca obiecte incluse mai multe coloane, numite în mod implicit **Column1**, **Column2**,... Modificarea unei proprietăţi sau a unei metode a unei coloane se face prin alegerea coloanei respective în lista derulantă din partea superioară a ferestrei proprietăţilor.*

O primă caracteristică a unei coloane este alinierea textului în cadrul celulelor sale, lucru stabilit prin intermediul proprietăţii **Alignmment**. Alinierea se poate face pe orizontală (valorile posibile fiind *Left* pentru stânga, *Center* pentru centru şi *Right* pentru dreapta) şi pe verticală (valorile valide fiind *Middle* pentru centru, *Top* pentru aliniere la latura superioară şi *Bottom* pentru aliniere la latura inferioară). O valoare suplimentară este *Automatic*, care determină stabilirea automată a modului de aliniere, în funcţie de tipul datelor din coloana respectivă.

O altă proprietate a unei coloane este **Movable**, care, dacă are valoarea *adevărat*, permite mutarea coloanei de către utilizator (la rulare) în altă poziţie. Mutarea se face prin tragerea cu mouse-ul a antetului coloanei în noua poziţie. Redimensionarea la rulare a coloanei este permisă numai dacă proprietatea **Resizable** are valoarea

adevărat. Operaţia se realizează prin tragerea cu mouse-ul a laturilor din stânga s dreapta ale coloanei respective.

Sursa de date a coloanei este stabilită prin intermediul proprietă **ControlSource**. În cazul unei grile care preia datele dintr-o tabelă, fiecare dint coloane are ca sursă de date un câmp al tabelei. Tipul de date al câmpului respec determină de obicei tipul obiectului de interfaţă conţinut în coloana grilei.

Sursa de date a coloanei este, în general, extinsă ca sursă de date şi pent obiectul de interfaţă conţinut în coloană. Aceasta înseamnă că, dacă o coloană pre datele dintr-un câmp al tabelei sursă, atunci câmpul de editare (de exemplu) conţinut coloană va avea ca sursă de date tot câmpul respectiv (va edita conţinutul câmpu tabelei). Extinderea sursei de date a coloanei la obiectele de interfaţă incluse în ea es dată de valoarea *adevărat* a proprietăţii **Bound**.

Parametrizarea antetului coloanelor grilei

Antetul unei coloane poate fi şi el parametrizat, în sensul că i se poate schim conţinutul, alinierea, fontul etc. De exemplu, fiecare coloană ar trebui să aibă ca an un text care să indice semnificaţia datelor editate în coloana respectivă. De asemene textul antetului ar putea fi afişat cu un font diferit de cel al coloanelor, de exemplu aldir

O primă proprietate a antetului unei coloane este **Caption**, care dă tex informativ afişat în antetul respectiv. Alinierea este stabilită prin intermediul proprietă **Alignment**, iar fontul folosit la afişarea textului este dat de proprietăţile de tip **Font...**

Parametrizarea obiectului de interfaţă inclus într-o coloană a grilei

O coloană conţine un obiect de interfaţă al cărui tip este stabilit în funcţie de tip de date al câmpului tabelei folosit ca sursă de date. La construirea unei grile coloane acesteia sunt „umplute" cu câmpuri de editare. Deseori însă, este necesar ca coloanele respective să fie introduse alte tipuri de obiecte, precum liste derulan comutatoare etc.

Modificarea tipului obiectului de interfaţă nu este posibilă direct, în cad Constructorului de forme. Există însă „artificii" cu ajutorul cărora se poate realiza ace lucru, o metodă fiind, de exemplu, modificarea manuală a tabelei în care este memora forma respectivă.

Să luăm, de exemplu, forma **Test**, în care a fost definită grila **Grid1**, în care dor să introducem în coloana a patra, **Column4**, un comutator. Pentru aceasta, vo deschide tabela `TEST.scx` în care sunt memorate datele referitoare la forma **Te** folosind comanda:

```
USE test.scx
```

Apoi deschidem o fereastră **Browse** pentru editarea acestei tabele. Comanda este, evident, BROWSE.

Test

Platform	Uniqueid	Timestamp	Class	Classloc	Baseclass	Objname	Pare
COMMENT	Screen		memo	memo	memo	memo	memo
WINDOWS	_SOFOSHARS	629369384	Memo	memo	Memo	Memo	memo
WINDOWS	_SOFOSHAUQ	629369940	Memo	memo	Memo	Memo	memo
WINDOWS	_SOFOSHAVH	629369384	Memo	memo	Memo	Memo	Mem
WINDOWS	_SOFOSHAVR	629369384	Memo	memo	Memo	Memo	Mem
WINDOWS	_SOFOSHAVT	629369384	Memo	memo	Memo	Memo	Mem
WINDOWS	_SOFOSHAVX	629369384	Memo	memo	Memo	Memo	Mem
WINDOWS	_SOFOSHAVZ	629369384	Memo	memo	Memo	Memo	Mem
WINDOWS	_SOFOSHAW3	629369384	Memo	memo	Memo	Memo	Mem
WINDOWS	_SOFOSHAW4	629369384	Memo	memo	Memo	Memo	Mem
WINDOWS	_SOFOSHAW8	629369384	Memo	memo	Memo	Memo	Mem
WINDOWS	_SOFOSHAW9	629369384	Memo	memo	Memo	Memo	Mem
WINDOWS	_SOFOSHAWD	629369384	Memo	memo	Memo	Memo	Mem
COMMENT	RESERVED		memo	memo	memo	memo	memo

În fereastra **Browse** se caută mai întâi linia care corespunde câmpului de editare ce se doreşte a fi înlocuit cu un comutator. Identificarea sa se face prin textul *Text1* conţinut în câmpul **Objname** (nume obiect) şi textul *Form1.Grid1.Column4* conţinut în câmpul **Parent** (părinte). Primul dintre câmpurile amintite indică numele obiectului formei, iar celălalt obiectul compus care îl conţine.

În această linie se fac următoarele modificări:

Conţinutul câmpului	se schimbă din	în	Observaţii
Class	`textbox`	`checkbox`	Clasa pa baza căreia se construieşte obiectul
Baseclass	`textbox`	`checkbox`	
Objname	`Text1`	`Check1`	Numele obiectului creat
Properties	...	golire	Proprietăţi specifice (se goleşte pentru a fi preluate valorile implicite)

O dată realizate aceste modificări, se închide tabela respectivă (închiderea ferestrei **Browse** şi comanda `CLOSE ALL` în fereastra de comenzi) şi se reporneşte Constructorul de ecrane, pentru a se efectua restul de modificări în mod interactiv.

Obs *Pentru a fi vizibil un obiect de interfaţă diferit de un câmp de editare, este necesară modificarea proprietăţii* ***Sparse*** *a coloanei respective din valoarea* adevărat *în* fals.

După stabilirea tipului de obiect de interfaţă din fiecare coloană, se trece la parametrizarea acestora, adică la stabilirea proprietăţilor şi a metodelor lor. Acest lucru depinde de tipul de obiect de interfaţă respectiv; pentru mai multe amănunte, a se vedea paragrafele anterioare.

Pagini alternative (Page Frame)

Aici este definit un set de pagini alternative, deoarece datele referitoare la o factură sunt foarte numeroase şi nu încap toate în formă în acelaşi timp

Factura

de marfa emisa

Numar 1 din 02/05/1998

Partener ALFA

Seria BDC13456801B

Nr.aviz insotire marfa 23421

Inregistrari | Termene | Plati | Contari | Expeditie

Cod	Denumire produs	U.M.	Cantitate	Pret unitar	Valoare	TVA
bere	Bere Gambrinus	buc	1240.00	2320.00	2876800.00	632896.00

Cantitatea din stoc din produsul selectat 0.00

Factura de retur | Reducere de pret 0.00 v | Total 2876800.00 632896.00

Total de plata 3509696.00

Anulata | Nom.Prod. | Nom.Part. | Salvare | Tiparire | Stergere | Terminare

Def

***Paginile alternative** reprezintă obiecte de interfaţă compuse, care se caracterizează prin faptul că permit definirea în aceeaşi zonă din formă a mai multor seturi de obiecte de interfaţă. Fiecare set de obiecte este inclus într-o pagină separată, iar paginile sunt afişate pe ecran alternativ (una singură la un moment dat).*

Paginile alternative se folosesc acolo unde numărul şi dimensiunea obiectelor de interfaţă ale unei forme fac ca acestea să nu încapă în forma respectivă. Obiectele vor fi grupate după diferite criterii şi plasate în mai multe pagini alternative.

Definirea unui set de pagini alternative debutează prin acţionarea butonului de pe bara utilitară a obiectelor de interfaţă. Ca şi în cazul celorlalte obiecte de interfaţă, urmează trasarea pe formă a zonei ce va fi ocupată de setul de pagini alternative şi modificarea proprietăţilor şi a metodelor acestuia pentru a corespunde aspectului şi comportamentului dorit.

O dată definit un set de pagini alternative, acesta trebuie configurat. Una dintre primele proprietăţi ce trebuie ajustate este **PageCount**, cu ajutorul căreia se specifică numărul paginilor care alcătuiesc setul respectiv.

Numărul indicat prin intermediul proprietăţii **PageCount** va da şi numărul de obiecte de tip pagină alternativă, **Page**, incluse în obiectul compus set de pagini alternative, **PageFrame**.

```
Set de pagini            PageFrame
 ├─► Pagină_1             ├─► Page_1
 └─► Pagină_2             └─► Page_2
     ...                      ...
```

Dimensiunile setului de pagini alternative în cadrul formei sunt date, evident, de proprietăţile **Left**, **Top**, **Width** şi **Height** (ca şi la celelalte obiecte de interfaţă). Dimensiunea verticală a unei pagini alternative (doar partea utilă, fără antet) este dată de proprietatea **PageHeight**, iar lăţimea de proprietatea **PageWidth**.

Antetul setului de pagini alternative este porţiunea superioară din care se selectează cu ajutorul mouse-ului pagina activă, adică cea în care se lucrează curent. În antetul fiecărei pagini este afişat un text informativ care indică semnificaţia grupului de obiecte de interfaţă incluse în pagina respectivă.

Prezenţa antetului setului de pagini alternative este stabilită de proprietatea **Tabs**. Dacă aceasta are valoarea *adevărat*, atunci pe ecran va fi afişat antetul paginilor, iar în caz contrar, zona rezervată antetului va fi ocupată de pagini (adică paginile se vor extinde pe toată suprafaţa obiectului).

Numărul paginilor unui set este variabil (la dispoziţia proiectantului), existând situaţii în care acest număr este mai mare (să zicem, peste 6, 7 pagini). În acest caz, în

La un moment dat, numai una dintre paginile unui set este activă, ea fiind indicată de proprietatea **ActivePage**.

O dată configurat obiectul set de pagini alternative, se poate trece la parametrizarea fiecărui obiect de tip pagină în parte. Pentru aceasta, trebuie mai întâi deschis pentru editare obiectul compus set de pagini, prin una dintre metodele următoare:

- se execută un clic cu butonul drept pe setul de pagini alternative şi, din meniul afişat pe ecran, se alege opţiunea **Edit**;
- se alege direct în fereastra de proprietăţi a Constructorului de forme pagina de modificat (Page1, Page2,...), ceea ce duce la afişarea proprietăţilor şi metodelor paginii respective.

Textul asociat unei pagini este dat de proprietatea **Caption** a paginii respective.

Exemplu

*De exemplu, pentru stabilirea prin cod a textului informativ asociat paginii a doua a setului de pagini alternative **PageFrame1**, se foloseşte comanda:*

```
PageFrame1.Page2.Caption="noul text"
```

Ordinea paginilor poate fi dată de proprietatea **PageOrder** (poziţie pagină) asociată fiecărei pagini. Această proprietate are valoarea 1 pentru prima pagină a setului, valoarea 2 pentru a doua pagină şi aşa mai departe.

Exemplu

*Dacă se doreşte ca pagina a cincea să devină pagina a doua, se modifică valoarea proprietăţii **PageOrder** din 5 în 2. Restul paginilor vor fi reordonate automat (pagina 2 va deveni 3, pagina 3 va deveni 4 şi aşa mai departe).*

Referirea la obiectele din cadrul unei pagini alternative a unui set se face fie cu construcţia:

```
PageFrame_x.Page_y.<obiect>.<proprietate sau metodă>
```

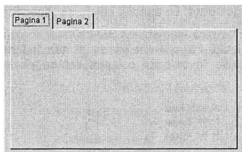
fie cu ajutorul unei proprietăţi speciale a obiectului de tip pagină, şi anume **Controls**. Această proprietate este un vector cu atâtea elemente câte obiecte de interfaţă sunt definite în pagina respectivă. Referirea la un obiect (la proprietăţile şi la metodele acestuia) din cadrul paginii se face astfel:

```
PageFramex.Pagey.Controls[<nr.obiect>].<proprietate sau metodă>
```

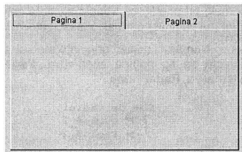
care nu mai pot fi afişate complet toate anteturile paginilor pe o singură linie, soluţia problemei este dată de proprietatea **TabStretch**. Această proprietate poate lua două valori, şi anume:

- *0 – Multiple Rows* (mai multe rânduri) – dacă sunt prea multe pagini şi anteturile lor nu încap pe o singură linie, atunci se creează două linii de anteturi;
- *1 – Single Row* (un singur rând) – se ajustează anteturile (chiar dacă textul acestora nu se mai vede complet), astfel încât toate să fie afişate pe o singură linie.

O altă proprietate referitoare la modul de afişare a anteturilor paginilor alternative este **TabStyle**. Ea stabileşte dacă anteturile paginilor alternative se vor întinde pe toată lăţimea obiectului de interfaţă (indiferent de dimensiunea textului informativ afişat în fiecare antet) – valoarea *1 – Nonjustified* – sau vor ocupa numai atât cât este necesar pentru a fi afişat tot textul informativ al anteturilor – cazul valorii *0 – Justified.*



Anteturile dimensionate după text



Anteturile pe toată lăţimea

Referirea în cod la obiectele componente ale unui set de pagini alternative se poate face cu ajutorul unei proprietăţi speciale, numită **Pages**. Aceasta reprezintă de fapt un vector care are ca elemente paginile componente ale setului. Prin urmare, modificarea oricărei proprietăţi a unei pagini (ca obiect conţinut în obiectul compus set de pagini alternative) se poate face printr-o construcţie de tipul:

```
<set pag.alt.>.Pages[<nr.pagină>].<proprietate>=<valoare nouă>
```

Exemplu

De exemplu, pentru modificarea textului informativ al paginii 2 a unui set de pagini alternative, se poate folosi construcţia:

```
PageFrame1.Pages[2].Caption="Text nou"
```

Ceasul (Timer)

Deseori, în lucrul cu un sistem informatic trebuie măsurate diferite intervale de timp şi declanşate anumite acţiuni la scurgerea acestor intervale. Să considerăm, de exemplu, afişarea pentru utilizator a unui mesaj de avertizare într-o procedură de prelucrare mai lungă. Mesajul respectiv trebuie afişat pe ecran, dar dacă utilizatorul nu este în faţa calculatorului, nu este necesară aşteptarea confirmării sale, ci se poate continua prelucrarea respectivă după un anumit interval de timp. Cu alte cuvinte, se consideră că utilizatorul a confirmat continuarea prelucrării, dacă, să zicem, timp de două minute nu a apăsat nici o tastă.

Pentru măsurarea acestor intervale de timp se foloseşte un obiect special de interfaţă, numit ceas.

Def ***Ceasul*** *reprezintă un obiect special de interfaţă, cu ajutorul căruia sunt măsurate intervalele de timp.*

Ceasul nu este un obiect de interfaţă vizibil (de exemplu, un ceas desenat pe formă), ci doar un mecanism care permite controlul timpului, mecanism implementat ca un obiect de interfaţă.

De fapt, definirea unui ceas în cadrul formei permite proiectantului specificarea unui interval de timp după care se declanşează un anumit eveniment. În metoda asociată acestui eveniment se pot introduce comenzi, care sunt executate numai după scurgerea timpului respectiv. Se poate controla momentul în care este pornit ceasul şi lungimea intervalului de timp.

Definirea unui ceas într-o formă se face cu ajutorul butonului de pe bara utilitară a obiectelor de interfaţă. După acţionarea acestui buton, un clic executat în formă determină plasarea ceasului în forma respectivă. Dimensiunea şi poziţia în formă a obiectului ceas nu este importantă, deoarece oricum acesta este invizibil la rulare.

Proprietatea ceasului care stabileşte intervalul de timp la care este generat evenimentul **Timer** este **Interval**. Această proprietate poate lua o valoare numerică, semnificând numărul de milisecunde după care este generat acest eveniment. Dacă valoarea proprietăţii **Interval** este 0, nu este generat nici un eveniment de tip **Timer** şi deci ceasul nu este activ.

După scurgerea fiecărui interval de timp (specificat prin **Interval**) este generat un eveniment **Timer**, eveniment care are ataşată metoda cu acelaşi nume. În această metodă pot fi incluse diferite comenzi, care să fie executate periodic.

Metoda **Reset** aduce contorul ceasului la 0, ceea ce face ca numărătoarea să reînceapă.

Exemplu

Un prim exemplu de folosire a ceasului este afişarea în cadrul formei a unei imagini, un interval de timp limitat, să zicem 5 secunde.

Pentru aceasta definim forma, în ea definim imaginea, iar în partea inferioară a formei construim un buton cu care se declanşează afişarea imaginii respective.



După definirea imaginii şi a butonului, mai definim în formă şi un ceas care va controla intervalul de timp respectiv.

*Iniţial, lăsăm imaginea invizibilă (proprietatea **Visible** are valoarea* fals*).*

*În metoda **Clic** a butonului introducem comanda de afişare a imaginii:*

```
ThisForm.Image1.Visible=.T.
```

şi pe cea de activare a ceasului:

```
ThisForm.Timer1.Interval=5000
```

*În metoda **Timer** a ceasului introducem comenzile ce trebuie executate după scurgerea intervalului de 5 secunde, adică cea pentru reascunderea imaginii:*

```
ThisForm.Image1.Visible=.F.
```

şi cea pentru dezactivarea ceasului:

```
This.Interval=0
```

Cu ajutorul ceasului se pot realiza diferite efecte dinamice, cum ar fi de exemplu cele de animație.

Exemplu

Am fi putut cere ca imaginea din exemplul anterior să fie afișată intermitent, la un interval de jumătate de secundă. Pentru aceasta, am fi stabilit intervalul de timp la 500 de milisecunde, prin comanda:

```
ThisForm.Timer1.Interval=500
```

*în metoda **Clic** a butonului.*

*În metoda **Timer** a ceasului nu am mai fi anulat intervalul de timp stabilit prin proprietatea **Interval**. De asemenea, starea imaginii, vizibilă sau invizibilă, ar trebui schimbată la fiecare interval de timp (în cea complementară), lucru realizat prin comanda:*

```
ThisForm.Image1.Visible=NOT(ThisForm.Image1.Visible)
```

Un efect de animație se poate obține prin mișcarea pe formă a unei imagini la diferite intervale de timp. De exemplu, schimbând poziția unei imagini la un interval de, să zicem, o zecime de secundă, obținem un efect de animație (ca și cum imaginea respectivă s-ar fi mișcat pe ecran).

Vă lăsăm dumneavoastră plăcerea de a realiza un astfel de test.

Bare utilitare (Toolbar)

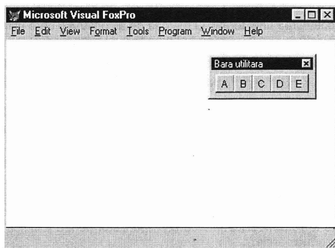
Def

***Barele utilitare** reprezintă tipuri speciale de forme, folosite pentru declanșarea rapidă a unor operații de prelucrare.*

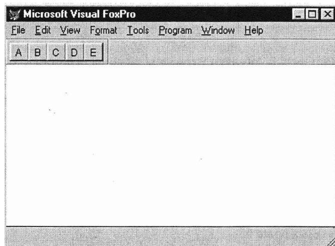
Barele utilitare sunt plasate de obicei în partea superioară a ferestrei aplicației, dar ele pot fi atașate și marginilor laterale ale ferestrei sau pot figura chiar ca ferestre independente plasate oriunde în cadrul ferestrei aplicației.

Un exemplu de bară utilitară este cea a obiectelor de interfață, prezentă în cadrul Constructorului de forme. Marea majoritate a sistemelor informatice (procesoare de texte, SGBD, sisteme de calcul tabelar etc.) conțin bare utilitare care ușurează foarte mult lucrul.

O bară utilitară poate apărea pe ecran ca o simplă formă, după cum se vede în figura următoare:



De asemenea, o bară utilitară poate fi ancorată la una dintre marginile ferestrei aplicației, operație realizată prin tragerea cu mouse-ul a barei peste marginea dorită. O dată atașată o bară utilitară la una dintre marginile ferestrei aplicației, zona utilă a acestei ferestre se micșorează. Cu alte cuvinte, o bară utilitară face parte din zona inutilă a ferestrei aplicației (ca și linia meniului, a titlului sau bara de stare din partea inferioară).



O bară utilitară este asemănătoare unei forme şi, de aceea, construirea ei se aseamănă cu construirea unei forme. O metodă de a construi o bară utilitară este aceea de a construi mai întâi o formă, care se transformă apoi în bară utilitară.

Să presupunem că dorim să construim bara utilitară de exerciţiu. Pentru aceasta, construim mai întâi o formă goală cu acelaşi nume. Comanda corespunzătoare este:

```
MODIFY FORM test
```

Salvăm forma respectivă, închidem Constructorul de forme şi apoi deschidem tabela în care este memorată forma. Pentru aceasta, folosim comanda:

```
USE test.scx
```

Intrăm în editarea tabelei `test.scx` cu ajutorul unei ferestre **Browse** (lansate prin comanda cu acelaşi nume, `BROWSE`). Linia a treia a tabelei va memora datele referitoare la formă, pe care le vom modifica în mod corespunzător pentru a indica o bară utilitară. Modificările sunt următoarele:

Conţinutul câmpului	se schimbă din	în	Observaţii
Class	`form`	`Toolbar`	Clasa pa baza căreia se construieşte obiectul
Baseclass	`form`	`Toolbar`	
Objname	`Form1`	`Toolbar1`	Numele obiectului creat
Properties	...	Golire	Proprietăţi specifice (se goleşte pentru a fi preluate valorile implicite)

Acum putem închide fereastra **Browse** şi tabela şi putem reveni la Constructorul de forme pentru a realiza restul configurărilor în mod interactiv.

Proprietăţile şi metodele barelor utilitare sunt asemănătoare cu cele ale formelor. Principalele diferenţe sunt cele legate de „ancorarea" barelor utilitare la marginile ferestrei aplicaţiei.

Ancorarea se poate face de către utilizator în timpul rulării sau poate fi stabilită apriori, de către proiectant. Metoda prin care se ancorează o bară utilitară este **Dock**. Aceasta se apelează cu un parametru ce indică locul ancorării:

```
<bară utilitară>.Dock(<poziţie>)
```

Parametrul `<poziţie>` poate fi:

- -1 – în cazul dezancorării;
- 0 – în cazul ancorării la marginea superioară a ferestrei aplicației;
- 1 – pentru ancorare la latura din stânga;
- 2 – pentru ancorare la latura din dreapta;
- 3 – pentru ancorare la latura de jos a ferestrei aplicației.

Pentru a afla dacă o bară utilitară este ancorată sau nu (este liberă în fereastra aplicației), se testează proprietatea **Docked** a acesteia. Valoarea *adevărat* indică ancorarea, iar valoarea *fals* indică dezancorarea.

Pentru a afla și locul ancorării, se folosește proprietatea **DockPosition**, a cărei valoare numerică are aceeași semnificație cu a parametrului metodei **Dock** (a se vedea mai sus).

În legătură cu ancorarea și dezancorarea unei bare utilitare există o serie de evenimente, care, interceptate, oferă posibilitatea executării anumitor comenzi specifice. Înaintea ancorării unei bare utilitare este generat un eveniment **BeforeDock**, iar după ancorare unul de tip **AfterDock**. Dezancorarea este însoțită de un eveniment **UnDock**.

Una dintre caracteristicile principale ale unei bare utilitare, spre deosebire de formă, este stabilirea automată a poziției obiectelor de interfață incluse în aceasta. Astfel, butoanele unei bare utilitare vor fi plasate unele lângă altele. La fel, pentru texte informative, comutatoare, liste derulante etc., se va calcula automat poziția lor pe bară, poziție care nu este stabilită manual de proiectant.

Plasarea unui obiect de interfață pe o bară utilitară este realizată prin apăsarea butonului corespunzător de pe bara utilitară a obiectelor de interfață, urmată de execuția unui clic în poziția în care se dorește plasarea obiectului. Proiectantul nu stabilește poziția exactă a obiectului pe bară, ci doar ordinea obiectelor respective. De exemplu, se poate pune un buton între alte două butoane plasate anterior pe bara utilitară, dar nu se.poate deplasa un buton puțin mai la stânga.

Există însă posibilitatea dimensionării obiectelor respective. De asemenea, există un obiect de interfață de un tip special, numit separator, care are rolul de a plasa între diferitele obiecte de interfață ale unei bare utilitare un spațiu (pentru gruparea logică a obiectelor respective). Plasarea unui separator într-o anumită poziție a barei utilitare se face prin acționarea butonului de pe bara utilitară a obiectelor de interfață, urmată de un clic simplu în locul unde se dorește plasarea acestuia.

Configurarea obiectelor de interfață ale unei bare utilitare se face la fel ca în cazul formelor (pentru mai multe informații, a se vedea paragrafele anterioare).

Tehnici speciale folosite la construirea formelor

Forme multiinstanță

Def **Formele multiinstanță** *sunt acele forme care permit rularea mai multor exemplare ale lor, chiar dacă sunt construite și memorate pe disc într-un singur exemplar.*

Def **Instanța** *unei forme reprezintă forma respectivă lansată în execuție. Dacă aceeași formă se lansează în execuție de mai multe ori, atunci avem de-a face cu mai multe instanțe ale aceleiași forme.*

Pentru a avea mai multe instanțe ale unei forme, este necesar ca la proiectarea formei să se țină cont de o serie de observații. Mai întâi, forma trebuie astfel proiectată încât să existe diferențiere între elementele specifice formei și cele specifice instanțelor acesteia.

Să luăm, de exemplu, cazul unei forme care, la construire, creează o tabelă temporară în care memorează diferite date. Lansarea în execuție a formei pentru prima oară nu ridică nici un fel de probleme. Dacă însă forma este lansată în execuție a doua oară (se creează a doua instanță a formei), atunci tabela temporară respectivă este creată a doua oară, iar dacă ea există, se inițializează (se golește de date). Dacă după crearea primei instanțe a formei au fost introduse date în tabela respectivă, atunci aceste date au fost pierdute la crearea celei de-a doua instanțe.

Aceeași problemă apare atunci când instanțe ale aceleiași forme lucrează cu variabile sau tablouri globale. De exemplu, ce se întâmplă când, la lansarea în execuție a unei forme, se inițializează o variabilă globală cu o valoare prestabilită? Această inițializare va avea loc la fiecare creare a unei noi instanțe a formei respective.

Soluția la astfel de probleme este dată de identificarea unică a elementelor specifice instanței. Un ajutor prețios în această direcție este oferit de sesiunile private de date ce pot fi atribuite unei forme (proprietatea **DataSession** a formei are valoarea *2 – Private Data Session*). Instanțele unei forme care are asociată o sesiune de date privată vor avea propriul lor mediu de lucru, fiecare instanță cu mediul său. În acest fel se realizează o independență sporită a formei respective, ea nemaidepinzând de numărul de lansări în execuție.

De exemplu, folosirea unei variabile globale într-o formă cu o sesiune de date privată face ca fiecare instanță a formei să lucreze cu o „copie" a variabilei globale respective, copie valabilă numai în mediul de date al instanței. Variabila respectivă va

dispărea atunci când instanța formei este distrusă, deoarece o dată cu instanța se elimină şi mediul ei de date.

În cazul tabelelor sau al altor tipuri de fişiere care sunt memorate pe disc, este necesară folosirea unei tehnici speciale, care să identifice în mod unic elementele specifice fiecărei instanțe în parte. De exemplu, dacă forma foloseşte o tabelă temporară, va trebui implementat un mecanism care să determine construirea unei tabele temporare pentru fiecare instanță a formei. De exemplu, tabela asociată primei instanțe a formei ar putea să se numească `TEMP_01`, cea asociată celei de-a doua instanțe `TEMP_02` şi aşa mai departe.

Identificarea unei instanțe a unei forme se poate face prin intermediul proprietății **DataSessionID**, care dă numărul sesiunii de date a formei respective. Dacă introducem acest număr în denumirea elementelor specifice instanțelor, atunci vom rezolva problema identificării unice a elementelor specifice instanțelor formei.

De exemplu, în loc ca tabela temporară să se numească `TEMP`, o vom numi `TEMP_`*`xx`*, în care *`xx`* va fi numărul sesiunii de date a formei, adică **DataSessionID**. Construcția prin care se obține acest lucru ar putea fi:

```
"TEMP_"+STRTRAN(STR(ThisForm.DataSessionID,2,0),' ','0')
```

Se observă că la şirul de caractere `"TEMP_"` se adaugă numărul sesiunii de date, transformat în şir de caractere. În construcția de mai sus s-a folosit funcția `STRTRAN()`, pentru a înlocui cu 0 eventualul spațiu la valorile mai mici decât 10.

Pe lângă elementele specifice fiecărei instanțe, există şi elemente specifice formei, independente de instanță. Astfel de elemente ar putea fi, de exemplu, variabile sau tabele, folosite în comun de toate instanțele formei respective.

De exemplu, o formă folosită pentru introducerea datelor într-o tabelă va folosi tabela respectivă ca element comun tuturor instanțelor sale, deoarece toate instanțele formei respective vor depune datele în acelaşi loc. Tabela respectivă va avea aceeaşi denumire în fiecare dintre instanțele formei, deci va putea fi inclusă direct în mediul de date al formei (pentru a fi deschisă automat la lansarea în execuție a formei).

De asemenea, comunicarea între diferitele instanțe ale aceleiaşi forme se face prin intermediul unor elemente comune (variabile globale în mediul de date implicit sau tabele).

Modificarea manuală a tabelei în care este memorată o formă

Una dintre tehnicile speciale folosite la realizarea diferitelor operații care nu sunt posibile direct din interiorul Constructorului de forme este modificarea manuală a tabelei în care este memorată forma respectivă. La rularea formei, această tabelă este interpretată de un modul special al sistemului, la execuția unei comenzi de tipul:

```
DO FORM <nume formă>
```

În tabela în care este memorată o formă, este rezervată câte o linie pentru fiecare element al formei. De exemplu, există o linie pentru mediul de date al formei, una pentru caracteristicile formei respective, una pentru fiecare obiect de interfață inclus în formă.

Pentru a afla care element este descris de o anumită linie a tabelei, se citesc câmpurile **Objname** (nume obiect) şi **Parent** (părinte). Primul dintre ele indică numele obiectului descris pe linia respectivă, iar cel de-al doilea succesiunea de obiecte care conțin elementul respectiv.

Exemplu

*De exemplu, dacă în câmpul **Objname** avem* `Text1`*, iar câmpul **Parent** conține* `Form1.Grid1.Column2`*, atunci pe linia respectivă este descris câmpul de editare **Text1**, conținut în coloana 2, **Column2**, a grilei **Grid1**, din forma **Form1**.*

Câmpul **Class** (şi **Baseclass**) arată clasa pe baza căreia este construit obiectul respectiv. De fapt, acest câmp dă tipul obiectului descris pe linia respectivă. În câteva exemple anterioare am transformat o formă (care are clasa `form`) într-o bară utilitară (inclusă în clasa `toolbar`).

În câmpul **Properties** (proprietăți) al tabelei sunt memorate proprietățile care diferă de cele implicite, cu următorul format:

```
<nume proprietate> = <valoare nouă>
```

Pentru metode se foloseşte câmpul **Methods** (metode), cu următoarea sintaxă:

```
PROCEDURE <nume metodă>
  <comenzi>
  ...
ENDPROC
```

Celelalte câmpuri ale tabelei ar trebui lăsate nemodificate, deoarece ele sunt folosite intern de către sistem.

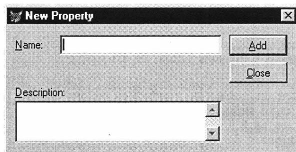
Adăugarea de proprietăți şi metode noi la o formă

O formă are o serie de proprietăți şi metode predefinite, dar îi pot fi ataşate şi proprietăți şi metode definite de utilizator. În proprietățile definite de utilizator se pot memora diferite valori necesare proiectantului pentru prelucrările realizate în metodele formei. De asemenea, la o formă pot fi adăugate şi tablouri, unidimensionale şi bidimensionale.

Proprietățile adăugate de utilizator sunt folosite atunci când se dorește realizarea unor transferuri de date între metodele unei forme. De exemplu, dacă într-o metodă a formei dorim să folosim o serie de parametri transmişi din exterior la apelarea ei, este necesară preluarea valorilor parametrilor transmişi metodei **Init** (acolo sunt receptați parametrii respectivi) în proprietăți ale formei, care să poată fi folosite ulterior în orice metodă a formei. În acest caz există şi varianta variabilelor globale, dar ea nu este eficientă în privința gestiunii memoriei şi a independenței modulelor respective.

Proiectantul poate adăuga la o formă şi metode proprii, prin intermediul cărora să realizeze diferite prelucrări. Aceste metode pot fi apelate din orice altă metodă a formei, predefinită sau adăugată de utilizator. Adăugarea metodelor la o formă sporeşte independența acesteia față de programul apelant, deoarece tot codul este memorat la un loc cu forma şi nu în biblioteci separate.

Adăugarea unei proprietăți la o formă se face în Constructorul de forme prin alegerea opțiunii **New Property...** (proprietate nouă) a meniului **Form**. În fereastra deschisă pe ecran se specifică numele noii proprietăți şi, eventual, o descriere a acesteia, după care se acționează butonul **Add** (adăugare). Fereastra se închide cu ajutorul butonului **Close**.



Referirea la o astfel de proprietate se face ca şi la proprietățile predefinite ale formei, adică printr-o construcție de tipul:

```
<formă>.<nume proprietate>
```

Dacă numele proprietății este precedat de paranteze pătrate care încadrează valori numerice, atunci proprietatea definită este de fapt un masiv.

Exemplu

De exemplu, `a[10]` *indică un vector de zece elemente numit* `a`*.*

Adăugarea unei metode la o formă se face cu ajutorul opţiunii **New Method...** (metodă nouă) a meniului **Form**. O dată adăugată o metodă, conţinutul acesteia trebuie specificat ca şi în cazul metodelor predefinite ale formei, adică în fereastra de cod a Constructorului de forme.

Folosirea Constructorilor pentru definirea formelor şi a obiectelor de interfaţă incluse în acestea

O tehnică specială folosită la construirea diferitelor elemente este aceea a Constructorilor sau a Vrăjitorilor. Visual FoxPro este foarte bogat în astfel de utilitare, pentru majoritatea elementelor dispunând de câte o astfel de unealtă.

Def

*Un **Constructor** sau un **Vrăjitor** este un utilitar al mediului care permite construirea rapidă şi simplă a unui anumit tip de element, prin specificarea principalelor proprietăţi ale acestuia în ferestre de dialog.*

O tehnică recomandată pentru construirea unui element (formă, obiect de interfaţă etc.) este folosirea constructorului corespunzător pentru definirea rapidă a elementului respectiv, urmată de modificarea parametrilor care vor fi diferiţi de cei stabiliţi implicit prin metoda automată a utilitarului.

Pentru pornirea vrăjitorului pentru forme (ca şi pentru alţi vrăjitori ai sistemului) se alege opţiunea **Wizards** a meniului **Tools**, iar din submeniul afişat se alege opţiunea **Form**.

Pentru definirea unui obiect de interfaţă cu ajutorul constructorului, în fereastra Constructorului de forme se execută un clic cu butonul drept pe obiectul de interfaţă respectiv (care trebuie să fi fost deja trasat în formă) şi, din meniul afişat, se alege opţiunea **Builder**. Urmează parcurgerea unei succesiuni de ferestre de dialog, în care se stabilesc diferite caracteristici ale obiectului respectiv.

Programe pentru raportare. Constructorul de rapoarte

Capitolul 10

- Introducere
- Constructorul de rapoarte – mod de folosire
- Proprietăţi ale raportului în ansamblul său
- Tipuri de elemente incluse în benzile rapoartelor
  - ✓ Texte informative (**Label**)
  - ✓ Câmpuri (**Field**)
  - ✓ Elemente semigrafice
  - ✓ Imagini
- Gruparea datelor în raport. Niveluri de grupare
- Folosirea variabilelor în construirea rapoartelor
- Rularea raportului. Afişarea pe ecran şi tipărirea

Introducere

Ce este un raport şi la ce foloseşte?

Faza finală a lucrului cu un sistem informatic este extragerea diferitelor informaţii pe baza datelor introduse anterior în bazele de date. Această fază este de fapt scopul, sau ţinta celorlalte operaţii (introducerea datelor, prelucrarea etc.).

Def

***Rapoartele** reprezintă ansambluri de informaţii construite pe baza datelor din tabele, informaţii care urmează a fi afişate pe ecranul monitorului sau tipărite la imprimantă.*

Deşi un raport se identifică de obicei cu forma sa externă (informaţiile propriu-zise), el reprezintă de fapt ceva mai mult, adică un ansamblu de elemente care conlucrează la selectarea şi furnizarea către utilizator a unui set de date într-o formă cât mai concisă, mai clară şi mai plăcută.

La nivelul sistemului de operare, un raport este un fişier special în care sunt memorate caracteristicile raportului şi ale elementelor sale componente. Acest fişier este interpretat de un modul special al SGBD la rularea raportului. Modulul preia datele din baza de date sursă şi le „aranjează pe foaie" în conformitate cu specificaţiile din fişierul raport respectiv. Situaţia obţinută este fie afişată pe ecran, fie tipărită la imprimantă.

Rapoartele sunt folosite atunci când este necesară informarea utilizatorului cu privire la anumite date din bazele de date. Ori de câte ori utilizatorul solicită consultarea unei baze de date, este necesară construirea unui raport cu datele respective.

Programele de raportare

Rapoartele sunt construite cu ajutorul programelor de raportare. În general, lucrul cu un astfel de program decurge în modul următor: mai întâi se intră într-o fereastră de dialog, în care utilizatorul precizează parametrii raportului. Acţionarea unui buton din această fereastră determină construirea şi afişarea pe ecran a raportului respectiv.

Exemplu

De exemplu, pentru construirea unui raport referitor la vânzările realizate într-o societate comercială de diferiţi agenţi, trebuie specificaţi o serie de parametri într-o fereastră de dialog de tipul următor:

RAPORT - Situatia vanzarilor

Anual	pentru anul	1999		
Lunar	pentru luna	Ianuarie	anul	1999
Interval	de la data	03/01/1999	pana la data	03/01/1999

Raport | Renuntare

Raportul propriu-zis este prezentat în paragraful următor.

Pornind de la acest scenariu, se poate pune în evidenţă structura generală a unui program de raportare. Acesta este format din trei secţiuni:

- una pentru preluarea de la utilizator a parametrilor necesari construirii raportului. În general, această secţiune constă dintr-o formă construită cu ajutorul Constructorului de forme;
- una pentru prelucrările datelor din bazele de date (extrageri, grupări, sortări etc.). De cele mai multe ori, bazele de date ale sistemului informatic nu au structura asemănătoare cu cea a raportului dorit, deoarece criteriile de proiectare a bazelor de date sunt optimizarea spaţiului pe disc, viteza de prelucrare şi altele, iar nu claritatea, lizibilitatea şi aspectul plăcut, caracteristici specifice rapoartelor. În general, această secţiune se ocupă cu extragerea datelor din bazele de date şi aducerea lor la o formă cât mai apropiată de cea a raportului (de obicei se creează o tabelă temporară, pe baza căreia va fi construit raportul respectiv);
- una pentru construirea formei exterioare a raportului. Acum, datele deja pregătite sunt preluate şi „aranjate" conform specificaţiilor proiectantului. Aceasta este o etapă de „cosmetizare" a datelor finale, astfel încât ele să capete un aspect plăcut, lizibil, clar.

În prezentul capitol ne vom ocupa în special de ultima dintre secţiunile de mai sus, adică de cea în care se stabileşte forma exterioară a raportului (fontul folosit, aranjarea datelor în pagină etc.).

Cum se construieşte şi cum se foloseşte un raport?

În Visual FoxPro distingem două moduri de creare a unui program de raportare:

- prin introducerea într-un program a tuturor comenzilor necesare construirii raportului (pentru culegerea parametrilor de la utilizator, pentru selectarea datelor din bazele de date şi pentru construirea efectivă şi afişarea raportului);
- prin introducerea într-o tabelă specială (cu extensia implicită .FRX) a tuturor caracteristicilor raportului, tabelă care să fie ulterior interpretată de un modul special al sistemului.

Prima variantă este cea clasică, folosită în versiunile mai vechi ale SGBD, în care uneltele interactive, vizuale, nu erau prezente. Din motive de compatibilitate, această variantă este disponibilă şi în Visual FoxPro, dar nu este recomandată pentru noile programe de raportare, deoarece necesită timp cunoştinţe şi eforturi suplimentare.

În cel de-al doilea caz, există două variante pentru construirea tabelei cu specificaţiile raportului: una manuală, prin comenzile FoxPro de prelucrare a tabelelor, şi alta automată, bazată pe folosirea unui utilitar special al sistemului, Constructorul de rapoarte.

Metoda manuală constă de fapt în crearea tabelei respective cu ajutorul comenzilor clasice din FoxPro. Dar pentru aceasta este necesară cunoaşterea structurii tabelei respective şi a detaliilor de codificare a caracteristicilor raportului şi ale elementelor acestuia.

Constructorul de rapoarte reprezintă un utilitar al sistemului, care permite utilizatorului construirea raportului în mod interactiv, prin folosirea diferitelor ferestre de dialog, opţiuni etc. Constructorul are grijă de toate detaliile de codificare şi memorare a specificaţiilor utilizatorului, acesta putându-se concentra asupra aspectelor importante, precum structura raportului, aspectul său etc.

Constructorul de rapoarte nu generează programul de raportare corespunzător, ci tabela care conţine datele raportului. Pentru construirea efectivă a raportului trebuie lansat în execuţie modulul care interpretează datele din tabelă, prin comanda REPORT FORM. Această comandă trebuie introdusă acolo unde se doreşte afişarea raportului, ca de exemplu în metoda **Click** a butonului **Raport** din forma de preluare a parametrilor raportului.

Prin urmare, un program de raportare ar putea fi construit în jurul unei forme. Aceasta urmează să conţină, pe lângă obiectele de interfaţă necesare citirii de la utilizator a parametrilor de rulare a raportului, şi un buton care declanşează construirea şi afişarea raportului (butonul **Raport**). La acţionarea acestui buton este declanşat atât programul de prelucrare/interogare a bazelor de date, care pregăteşte datele de afişat în raport, cât şi comanda de afişare propriu-zisă a raportului (REPORT FORM).

Structura unui raport cu niveluri de grupare. Exemplu

Structura rapoartelor poate diferi foarte mult de la caz la caz, de la simple prezentări ale conţinutului uneia sau mai multor tabele, la situaţii complexe, cu configuraţii neregulate. Un tip special de raport este acela cu niveluri de grupare, de totalizare. Aceste rapoarte reprezintă afişări ale unei baze de date în care înregistrările sunt grupate după diferite criterii, pentru fiecare grup în parte putând fi generată o înregistrare totalizatoare. În această linie totalizatoare se prezintă valoarea unui anumit câmp, cumulată pentru întregul grup.

Un exemplu de raport cu două niveluri de totalizare este *„Situaţia vânzărilor lunare pe persoane"*.

Titlu raport → **Situatia lunara a vanzarilor**

	Data	Nume	Prenume	Functia	Valoarea
Antet nivel 1	**Luna Ianuarie**				
Antet nivel 2	**Data**	**Nume**	**Prenume**	**Functia**	**Valoarea**
	05/01/1998	Georgescu	Marian	Agent	260000
	06/01/1998				130000
	09/01/1998				1345000
Concluzii nivel 2				**Total persoana in luna Ianuarie**	1735000
	Data	**Nume**	**Prenume**	**Functia**	**Valoarea**
	02/01/1998	Popescu	Mihai	Vanzator	110000
	07/01/1998				60000
	16/01/1998				415000
	23/01/1998				235000
Concluzii nivel 2				**Total persoana in luna Ianuarie**	820000
	Data	**Nume**	**Prenume**	**Functia**	**Valoarea**
	10/01/1998	Iancu	Dumitru	Agent prim	120000
	14/01/1998				460000
	27/01/1998				240000
	27/01/1998				1560000
Concluzii nivel 2				**Total persoana in luna Ianuarie**	2380000
			...		
Concluzii nivel 1				**Total luna Ianuarie pentru tot personalul**	7445000
Restul raportului			...		
				Total luna Decembrie pentru tot personalul	17860000
Concluzii raport				**Total anual pentru tot personalul**	118576000

Obs

Construirea raportului prezentat anterior s-a realizat pornind de la tabela vânzărilor efectuate în cadrul unei unităţi economice într-un an, tabelă cu următoarea structură:

DATA	D	8	**data efectuării vânzării**
NUME	C	12	**numele persoanei**
PRENUME	C	20	**prenumele persoanei**
FUNCTIA	N	2	**funcţia persoanei respective**
VALOAREA	N	12	**valoarea vânzării**

Observăm în acest raport (pentru care nu ne interesează momentan modul de obţinere) că pentru fiecare lună sunt prezentate vânzările fiecărui angajat în parte, cu un total lunar pentru fiecare angajat şi cu totalul lunar pentru toţi angajaţii. Primul nivel de totalizare este pe luni, adică cel exterior. În cadrul fiecărei grupe de înregistrări (adică înregistrările dintr-o anumită lună) se realizează o nouă grupare, după numele şi prenumele angajatului, obţinându-se astfel cel de-al doilea nivel de totalizare (nivelul interior). S-ar putea realiza încă un nivel de totalizare, de exemplu pe decade în cadrul fiecărei luni, pentru fiecare angajat în parte, nivel ce ar fi numerotat cu trei.

După cum s-a observat în exemplul anterior, nivelurile de totalizare se obţin prin gruparea înregistrărilor tabelei după anumite criterii. Fiecare nivel de totalizare corespunde unui criteriu de grupare, caracterizat la rândul lui de o cheie de grupare. În exemplul nostru, primul nivel de totalizare corespunde grupării pe luni, cheia fiind luna, iar celui de-al doilea nivel de totalizare îi corespunde gruparea după nume şi prenume (deci după persoană), cheia de grupare fiind dată de numele şi prenumele persoanei.

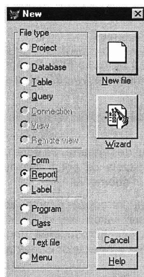
Constructorul de rapoarte – mod de folosire

Cum se porneşte Constructorul de rapoarte?

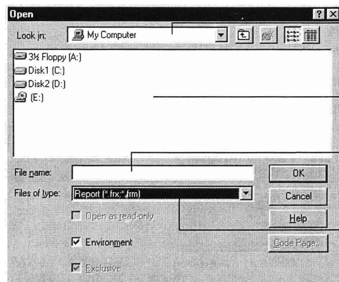
Pentru pornirea Constructorului de rapoarte se poate folosi fie comanda:

```
CREATE REPORT <nume raport>
```

introdusă în fereastra de comenzi, fie metoda interactivă. Aceasta din urmă constă în alegerea opţiunii **New** (nou) a meniului **File** şi alegerea butonului **Report** (raport) din fereastra de dialog afişată pe ecran, indicându-se astfel tipul de element ce urmează a fi creat.



După acționarea butonului **New file** (fişier nou) urmează precizarea numelui raportului într-o fereastră de dialog specifică.



Aici se stabileşte directorul din care se preia raportul

De aici se selectează raportul de modificat

Numele raportului se poate introduce direct în acest câmp de editare

Din această listă se alege tipul elementului de modificat ***(Report** – raport)*

Modificarea unui raport creat anterior se face cu ajutorul comenzii:

```
MODIFY REPORT <nume raport>
```

sau prin alegerea opţiunii **Open** a meniului **File**. Pe ecran va fi deschisă o fereastră în care se va selecta raportul ce urmează a fi modificat (a se vedea figura anterioară).

Elementele Constructorului de rapoarte

O dată pornit Constructorul de rapoarte, pe ecran este afişată fereastra de lucru a acestuia şi bara utilitară cu obiectele ce pot fi introduse în benzile raportului, iar la meniul sistemului este adăugat un submeniu.

În cadrul benzilor trebuie introduse diferite tipuri de obiecte, precum texte informative, câmpuri ale tabelelor, valori ale variabilelor, elemente semigrafice (linii, chenare etc.) sau grafice (imagini). Introducerea unui astfel de obiect în raport se realizează cu ajutorul barei utilitare a obiectelor Constructorului de rapoarte:



Operaţia constă, de fapt, în acţionarea butonului corespunzător de pe bara utilitară şi trasarea pe raport, în banda dorită, a zonei ce va fi ocupată de obiect. Urmează configurarea obiectului introdus.

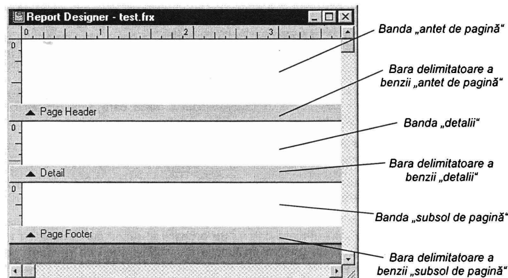
De asemenea, la pornirea Constructorului de rapoarte, este adăugat la meniul sistemului un submeniu care conţine opţiuni referitoare la diferite operaţii. Meniul se numeşte **Report** şi arată astfel:



Semnificaţia opţiunilor meniului va fi dezvăluită pe măsură ce vor fi tratate subiectele corespunzătoare.

Fereastra de lucru a Constructorului de rapoarte. Benzile unui raport

O dată pornit Constructorul de rapoarte, pe ecran este deschisă fereastra acestuia, care arată astfel:



Un raport construit cu Constructorul de rapoarte este alcătuit din benzi; în figura de mai sus sunt vizibile trei: **Page Header** (antet de pagină), **Detail** (detalii) şi **Page Footer** (subsolul paginii). Fiecare bandă va genera în raportul afişat mai multe **instanţe** ale sale, în funcţie de semnificaţia ei. Cu alte cuvinte, o bandă a unui raport reprezintă o formă generică a unei anumite zone din raportul final. Banda nu redă exact ceea ce va fi afişat în raportul final, ci în ea este codificat modul în care se determină ceea ce se afişează în raport.

Exemplu

*De exemplu, banda antetului de pagină (**Page Header**) se foloseşte pentru specificarea conţinutului părţii superioare a fiecărei pagini a raportului. Această bandă va genera în raportul final, pe fiecare pagină, câte o zonă care va cuprinde o serie de informaţii (în funcţie de conţinutul benzii).*

Banda antetului de pagină are atâtea instanţe câte pagini va avea raportul. Ea poate conţine, de exemplu, numărul curent al paginii, care va fi diferit de la o pagină a raportului la alta. În banda antet de pagină se va introduce o variabilă care indică modul de determinare a numărului respectiv.

În Visual FoxPro, un raport poate conţine următoarele benzi:

- **Detail** (detalii) – această bandă va genera în raport rândurile de detalii (de fapt, conţinutul de bază al raportului). Pentru un raport care preia datele dintr-o singură tabelă, banda de detalii a raportului corespunde înregistrărilor tabelei;
- **Title** (titlu) – banda corespunde antetului raportului final, adică ea generează în raportul final o zonă afişată o singură dată, la începutul raportului (pe prima pagină a acestuia). În această bandă se introduce de obicei titlul raportului şi eventual, capul de tabel, dacă acesta nu trebuie repetat pe fiecare pagină în parte);
- **Summary** (rezumat) – se foloseşte pentru specificarea zonei de sfârşit a raportului. Banda generează în raportul final o zonă care este afişată pe ultima pagină a raportului, la sfârşit, o singură dată;
- **Page Header** (antet de pagină) – în această bandă se specifică textul care va fi afişat în partea superioară a fiecărei pagini a raportului final;
- **Page Footer** (subsol de pagină) – cu ajutorul acestei benzi se precizează conţinutul părţii inferioare a fiecărei pagini a raportului final;
- **Group Header *n*** (antetul grupului de nivel *n*) şi **Group Footer *n*** (subsolul grupului de nivel *n*) – fiecare criteriu de grupare al raportului generează în raportul final mai multe grupuri, în funcţie de valorile cheii de grupare. La începutul fiecărui grup de nivel *n* va fi afişat conţinutul benzii antet de grup, iar la sfârşitul fiecărui grup conţinutul benzii subsol de grup;
- **Column Header** (antetul de coloană) şi **Column Footer** (subsolul de coloană) – la nivel de coloană, poate fi specificat un antet şi un subsol, fiecăruia corespunzându-i în fereastra de lucru a Constructorului câte o bandă.

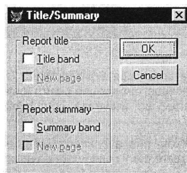
Nu toate benzile apar la pornirea Constructorului, dar ele pot fi afişate ulterior prin alegerea unor opţiuni. Construirea unui raport în Constructorul de rapoarte înseamnă, de fapt, specificarea benzilor care compun raportul, a dimensiunilor şi a conţinutului acestora.

O bandă a unui raport are asociată o bară (a se vedea figura anterioară), care o delimitează pe verticală şi cu ajutorul căreia poate fi redimensionată. Dimensiunea verticală a unei benzi corespunde dimensiunii verticale a zonei respective din pagina raportului final. Schimbarea dimensiunii unei benzi se face cu ajutorul mouse-ului, prin tragerea barei delimitatoare asociate.

La limită, o bandă poate avea înălţimea zero, ceea ce înseamnă că ea nu va apărea în pagina raportului final, deci nu va genera nimic în acesta. Pentru ca o bandă să aibă dimensiunea zero, bara sa trebuie alipită de bara benzii anterioare.

Un raport conţine cel puţin trei benzi: a antetului de pagină, a detaliilor şi a subsolului de pagină. Acestea fac parte oricum din raport. Dacă se doreşte inhibarea afişării lor, trebuie reduse la dimensiunea zero (chiar dacă apar în fereastra raportului, nu generează nimic în raportul final).

Benzile de titlu şi de concluzii sunt afişate prin alegerea opţiunii **Title/Summary** (titlu/rezumat) a meniului **Report**, pentru deschiderea următoarei ferestre de dialog:

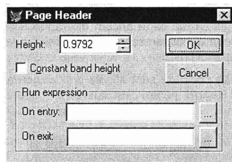


Activarea comutatorului **Title band** (banda de titlu) determină afişarea în raport a benzii titlului raportului. Pentru banda concluziilor raportului comutatorul este **Summary band** (banda rezumat). Fiecare dintre aceste două comutatoare are asociat un alt comutator, **New page** (pagină nouă), la activarea căruia banda corespunzătoare va începe pe pagină nouă.

Benzile de antet de grup şi subsol de grup sunt afişate în raport atunci când sunt definite criteriile de grupare respective (a se vedea paragraful din acest capitol referitor la gruparea datelor). Benzile de antet de coloană şi subsol de coloană sunt afişate numai în cazul rapoartelor cu mai multe coloane.

Proprietăţile benzilor raportului

O bandă a unui raport are o serie de proprietăţi care pot fi modificate de utilizator. Executarea unui clic dublu pe bara delimitatoare a unei benzi determină deschiderea pe ecran a unei ferestre de tipul celei de mai jos:



Dimensiunea exactă a benzii poate fi specificată în câmpul de editare **Height** (înălţime). Dacă se doreşte aceeaşi dimensiune pentru toate instanţele benzii, indiferent de conţinutul fiecărei instanţe, se activează comutatorul **Constant band height** (înălţime constantă a benzii).

Conform modelului dirijării prin evenimente, afişarea instanţei unei benzi este însoţită de două evenimente, care pot fi configurate de utilizator. Acestea sunt:

- înainte de fiecare afişare a benzii – cu ocazia acestui eveniment poate fi evaluată o anumită expresie, specificată în câmpul **On entry** (la intrare) din secţiunea **Run expression** (expresia de rulat);
- după fiecare afişare a benzii – cu ocazia acestui eveniment poate, de asemenea, să fie evaluată o anumită expresie, care va fi introdusă în câmpul de editare **On exit** (la ieşire).

Expresiile evaluate la apariţia acestor evenimente pot cuprinde funcţii definite de utilizator, care, la rândul lor, pot conţine diferite instrucţiuni. În acest fel pot fi executate diferite comenzi înainte sau după afişarea unei benzi, fiind astfel posibilă obţinerea unor efecte deosebite.

Exemplu

De exemplu, pentru întreţinerea unui contor extern în care să se memoreze numărul de pagini tipărite, se poate executa o instrucţiune de incrementare a contorului înainte de (sau după) fiecare afişare a benzii antet de pagină a raportului. Instrucţiunea respectivă trebuie introdusă într-o funcţie definită de utilizator, al cărei apel se introduce în expresia ***On entry*** *sau* ***On exit*** *a benzii.*

Proprietăți ale raportului în ansamblul său

Pentru exemplificarea modului de lucru cu Constructorul de rapoarte, în paralel cu prezentarea care urmează în acest capitol, vom construi un exemplu de raport, și anume *„Situația mijloacelor fixe ale unității economice, pe categorii "*. Varianta finală a acestuia arată ca în figura de mai jos:



Situatia mijloacelor fixe
ale unitatii economice pe categorii

Data curenta *10/01/1999*

Categoria *Tehnica de calcul*

Cod	**Denumire**	**Val. inventar**	**Val. amortizata**
CPII266	*Calculator Pentium II 266 MHz (1)*	*10234000*	*5467000*
CPDX66	*Calculator 486DX2 66MHz (4)*	*7450000*	*3245000*
CPMMX166	*Calculator Pentium MMX 166 MHz (7)*	*9254000*	*1134000*
IHPLJ5L	*Imprimanta A4 HP LaserJet 5L (1)*	*4560000*	*2300000*
IELX600	*Imprimanta A3 Epson LX600 (2)*	*1879000*	*7893000*
SCGCPV+	*Scaner Genius ColorPage Vivid+ (2)*	*1120000*	*456000*
	Total categorie	*34497000*	*20495000*

Categoria *Masini de transport persoane*

Cod	**Denumire**	**Val. inventar**	**Val. amortizata**
	Total categorie	*43567000*	*24598000*
	Total unitate	*342046000*	*154678000*

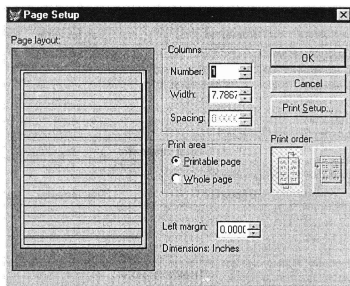
Pagina *16*

Pentru economie de spaţiu, s-a prezentat numai o parte din raport (începutul şi sfârşitul), datorită faptului că originalul se întinde pe 16 pagini.

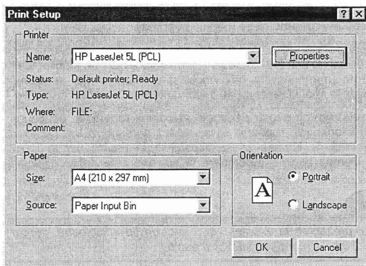
Pentru construirea raportului se porneşte de la o bază de date formată din două tabele: una numită **MIJFIX**, care conţine date referitoare la mijloacele fixe ale unităţii, şi alta numită **CATEG**, în care sunt descrise categoriile de mijloace fixe. Cele două tabele sunt legate între ele prin codul **categorie**.

Formatul paginii

Prima caracteristică ce trebuie configurată la construirea unui raport este formatul paginii pe care va fi tipărit raportul. Această proprietate se stabileşte în fereastra **Page Setup** (parametri pagină), deschisă prin alegerea opţiunii cu acelaşi nume a meniului **File**.



Dimensiunea fizică a paginii pe care se tipăreşte raportul este dependentă de imprimanta cuplată la sistemul de calcul şi se stabileşte în fereastra de configurare a imprimantei, **Print Setup**.



Pentru a deschide această fereastră se apasă butonul **Print Setup** (parametri imprimantă) din fereastra **Page Setup**.

Imprimanta se alege din lista derulantă **Name** (nume) din secţiunea **Printer** (imprimantă). Lista derulantă **Size** (dimensiune) stabileşte dimensiunea paginii, iar **Source** (sursă) sursa de hârtie a imprimantei. Orientarea paginii este dată de butoanele de selecţie **Portrait** (portret) – aşezare pe verticală – şi **Landscape** (peisaj) – aşezare pe orizontală. Închiderea ferestrei **Print Setup** determină revenirea în fereastra **Page Setup**.

Exemplu

În cadrul exemplului nostru, vom folosi o pagină A4 şi o imprimantă de tipul HP LaserJet 5L. Orientarea paginii va fi de tip portret.

După specificarea formatului fizic al paginii, este necesară specificarea formatului logic al paginii, adică a acelor proprietăţi care stabilesc zona efectivă în care este afişat raportul. Aceste proprietăţi sunt controlate cu ajutorul ferestrei **Page Setup**, astfel:

- numărul de coloane este stabilit în câmpul **Number** (număr), lăţimea unei coloane în câmpul **Width** (lăţime), iar spaţiul dintre coloane în câmpul **Spacing** (spaţiu);
- în general, imprimantele rezervă o zonă la marginea paginii de hârtie în care nu se realizează nici o tipărire. Pentru construirea raportului se poate folosi întreaga pagină fizică de hârtie, atunci când se selectează butonul **Whole**

page (întreaga pagină), sau doar zona tipăribilă a paginii, în cazul selectării butonului **Printable page** (pagina tipăribilă).

- în partea stângă a paginii de hârtie se poate lăsa un spaţiu (folosit la prinderea paginii într-un dosar, de exemplu) care se stabileşte în câmpul **Left margin** (marginea stângă). Acest spaţiu suplimentar se adaugă, în stânga paginii, la spaţiul rezervat de imprimantă.

O dată stabiliţi toţi aceşti parametri ai paginii raportului, conţinutul ferestrei de lucru a Constructorului de rapoarte se ajustează corespunzător. De exemplu, lăţimea benzilor va fi dată de lăţimea zonei disponibile în pagină, iar în cazul unui raport multicoloană, lăţimea benzilor referitoare la coloane este dată de lăţimea coloanelor respective.

Exemplu

Raportul prezentat ca exemplu va fi unul de tip unicoloană, pentru care vom rezerva 2 cm în marginea stângă a paginii.

Mediul de date al raportului

Un raport preia date din una sau mai multe tabele şi le aranjează în pagină, deci la construirea raportului este necesară specificarea tabelelor sursă. Acestea împreună cu anumite variabile folosite de raport alcătuiesc „mediul de date" al raportului, care este un obiect în sensul POO (deci are ataşate proprietăţi şi metode).

Mediul de date al unui raport se configurează în fereastra mediului de date, deschisă ca urmare a alegerii opţiunii **Data Environment** a meniului **View**. O dată deschisă această fereastră, la meniul sistemului este adăugat un nou submeniu, numit **DataEnvironment** (mediu de date), care conţine opţiuni referitoare la configurarea mediului de date. Datorită faptului că mediul de date reprezintă un obiect, la afişarea acestei ferestre devin disponibile şi uneltele de manipulare a proprietăţilor şi a metodelor mediului de date (a se consulta capitolul referitor la Constructorul de ecrane pentru a afla modul de lucru cu fereastra de proprietăţi şi fereastra de cod).

În mediul de date al unui raport pot fi specificate tabelele sursă, ceea ce are ca efect deschiderea automată a acestora la rularea raportului (fără a mai fi necesare instrucţiuni speciale în acest sens).

Adăugarea unei tabele la mediul de date al unui raport se face prin alegerea opţiunii **Add** (adăugare) a meniului **DataEnvironment**; pe ecran se deschide o fereastră din care proiectantul poate alege tabela dorită. Dacă tabela este legată de alte tabele în cadrul unei baze de date, în mediul de date al raportului se vor încărca şi legăturile respective (care, de asemenea, vor fi construite automat la rularea raportului).

Obs

În cazul unui raport care preia date din două sau mai multe tabele, este necesară specificarea în mediul de date al raportului şi a legăturilor între tabele, pentru ca acestea să fie parcurse în mod corect. De exemplu, dacă mediul de date al unui raport conţine două tabele între care nu s-au stabilit relaţii, iar în banda de detalii a raportului sunt introduse câmpuri ale ambelor tabele, după rulare, în raportul final va apărea câte o linie pentru fiecare înregistrare a celor două tabele, ceea ce nu este corect (ar trebui ca în raport să apară câte o linie pentru fiecare înregistrare părinte, completată cu datele din înregistrarea copil corespunzătoare).

Dacă se doreşte specificarea dinamică (la rulare) a tabelelor sursă, atunci se pot folosi metodele mediului de date, în care se vor introduce instrucţiunile corespunzătoare de deschidere a tabelelor.

Principalele metode ale mediului de date, asociate diferitelor evenimente, sunt:

- **BeforeOpenTables** – înainte de deschiderea tabelelor – aici se pot stabili dinamic numele şi locaţia tabelelor ce urmează a fi deschise;
- **OpenTables** – deschiderea tabelelor – aici se includ comenzile pentru deschiderea efectivă a tabelelor;
- **AfterOpenTables** – după deschiderea tabelelor – în această metodă se poate selecta tabela curentă şi înregistrările curente;
- **BeforeCloseTables** – înainte de închiderea tabelelor – se pot efectua calcule statistice asupra tabelelor modificate în formă;
- **CloseTables** – închiderea efectivă a tabelelor – se închid efectiv tabelele formei;
- **AfterCloseTables** – după închiderea tabelelor;

Principalele proprietăţi ale mediului de date sunt următoarele:

- **InitialSelectedAlias** – stabileşte tabela care va fi selectată iniţial;
- **AutoOpenTables** – determină, în cazul valorii logice `.T.`, deschiderea automată a tabelelor specificate în mediul de date al formei;
- **AutoCloseTables** – determină, în cazul valorii logice `.T.`, închiderea automată a tabelelor specificate în mediul de date al formei.

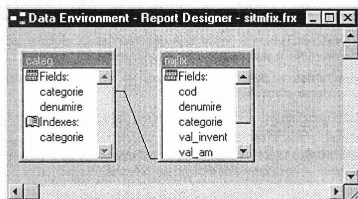
Obs

O dată specificate tabelele sursă în mediul de date al raportului, câmpurile acestora sunt disponibile pentru selectare şi în celelalte ferestre folosite la configurarea raportului. De exemplu, la definirea unui câmp într-o bandă a raportului, sursa câmpului poate fi selectată direct dintre câmpurile tabelelor sursă.

Ca şi în cazul formelor, mediul de date al unui raport poate fi unul privat sau cel implicit al sistemului. Avantajul unui mediu de date privat este independenţa faţă de condiţiile în care se face apelarea raportului, motiv pentru care se preferă această variantă. Pentru ca un raport să aibă un mediu de date privat, trebuie activată opţiunea **Private Data Session** (sesiune de date privată) a meniului **Report**.

Exemplu

Mediul de date al raportului prezentat ca exemplu („Situaţia mijloacelor fixe ale unităţii economice") conţine două tabele: `CATEG` *(pentru categoriile de mijloace fixe) şi* `MIJFIX` *(pentru datele referitoare la mijloacele fixe ale unităţii), legate între ele printr-o relaţie de tip „una-la-mai-multe".*



Fontul implicit al raportului

O altă caracteristică a unui raport este fontul implicit. Ea este folosită la proiectarea raportului, în sensul că fiecare obiect nou definit în raport va avea fontul implicit al raportului. Cu toate acestea, fontul unui obiect poate fi schimbat ulterior.

Prin urmare, la construirea unui raport, trebuie stabilit ca font implicit cel care predomină printre obiectele raportului, pentru celelalte obiecte fontul urmând a fi specificat explicit.

Stabilirea fontului implicit al raportului se face prin alegerea opţiunii **Default Font** (font implicit) a meniului **Report**, iar caracteristicile fontului sunt specificate în fereastra de dialog deschisă pe ecran.

Tipuri de elemente incluse în benzile rapoartelor

Texte informative (Label)

Unul dintre principalele elemente care pot fi incluse într-o bandă este textul informativ. Acesta reprezintă un şir de caractere invariabil pentru toate instanţele benzii respective în raportul final. În general, titlul unui raport reprezintă un text informativ, deoarece el este acelaşi indiferent de conţinutul tabelelor sursă.

Exemplu

De exemplu, titlul raportului prezentat este un text informativ. De asemenea, texte informative sunt şi „Data curenta", „Categoria", „Cod", „Denumire" etc.

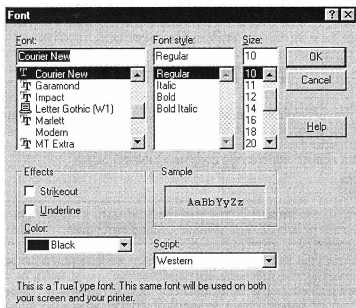
Introducerea unui text informativ într-un raport se face prin acţionarea butonului A de pe bara utilitară a obiectelor Constructorului de rapoarte şi plasarea cursorului (printr-un clic) în poziţia noului obiect, în banda dorită. Urmează specificarea conţinutului obiectului, adică scrierea textului respectiv. Când se termină introducerea textului, se execută un clic în afara obiectului nou creat. Din acest moment, obiectul devine un tot, următoarele caracteristici precizate de proiectant referindu-se la întregul text.

Obs

Dacă dorim ca o porţiune a unui text informativ să aibă caracteristici diferite (de exemplu altă dimensiune a fontului), este necesară împărţirea textului în mai multe obiecte de tip text informativ.

După specificarea conţinutului urmează configurarea obiectului creat, adică stabilirea poziţiei exacte şi a fontului folosit (set de caractere, dimensiune, culoare etc.). Poziţia se poate indica prin tragerea cu ajutorul mouse-ului pe suprafaţa raportului. Se pot efectua şi trageri dintr-o bandă în alta, nu numai în cadrul aceleiaşi benzi.

Fontul folosit la afişarea textului este specificat cu ajutorul ferestrei **Font**, care se deschide la alegerea opţiunii cu acelaşi nume a meniului **Format**.



În această fereastră pot fi specificate toate caracteristicile fontului, adică setul de caractere (lista **Font**), dimensiunea (lista **Size**), îngroşarea sau înclinarea (lista **Font style**), sublinierea (comutatorul **Underline**), tăierea cu o linie orizontală (comutatorul **Strikeout**), culoarea (lista **Color**).

Câmpuri (Field)

Câmpurile reprezintă elementele variabile ale unui raport. Cu alte cuvinte, un câmp inclus într-o bandă a raportului va genera în raportul final mai multe valori (în funcţie de numărul de instanţe ale benzii respective în raportul final), iar valorile vor fi determinate de sursa de date a câmpului respectiv.

Exemplu

Exemple de câmpuri sunt data curentă afişată în antetul raportului, codul mijlocului fix, denumirea şi valoarea mijlocului fix, numărul de pagină.

De cele mai multe ori, un câmp al raportului are ca sursă de date fie un câmp al unei tabele sursă, fie o variabilă a raportului. Pentru fiecare instanţă a benzii respective va fi calculată valoarea câmpului tabelei sau a variabilei respective şi, în raportul final, va fi afişată valoarea obţinută. Ca sursă de date a unui câmp poate fi specificată şi o

expresie, care va fi evaluată la rularea raportului pentru fiecare instanţă a benzii care conţine câmpul.

Un câmp al unui raport se caracterizează prin:

- sursa de date, adică modul în care se calculează valorile afişate pe poziţia câmpului respectiv în forma finală a raportului;
- formatul de afişare a valorii din câmp. Acest format este asemănător cu formatul specificat pentru câmpurile de editare din forme;
- funcţia de calcul aplicată valorii din câmp (însumare, medie etc.);
- condiţii dinamice de afişare a câmpului respectiv. Acestea se specifică atunci când se doreşte ca un câmp să fie afişat doar dacă sunt îndeplinite anumite condiţii.

Definirea unui câmp debutează prin acţionarea butonului de pe bara utilitară a obiectelor Constructorului de rapoarte, urmată de un clic simplu în locul unde va fi introdus noul obiect. După acest clic, pe ecran se deschide o fereastră de dialog:

Aici utilizatorul poate introduce parametrii câmpului. În câmpul de editare **Expression** (expresie) al acestei ferestre se poate indica sursa de date a câmpului, adică o expresie care va fi evaluată pentru fiecare instanţă a benzii respective. De multe ori, această expresie este un simplu câmp al unei tabele sursă sau o variabilă globală

sau locală raportului. Dar ea poate fi şi o expresie complexă, conţinând câmpuri ale tabelelor sursă, variabile, constante, funcţii ale limbajului etc. Expresia se poate introduce şi cu ajutorul Constructorului de expresii, pornit la acţionarea butonului din dreapta câmpului de editare **Expression**.

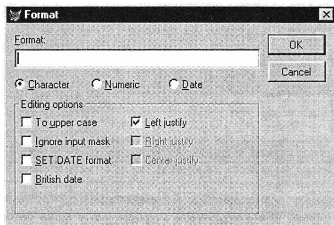
Exemplu

De exemplu, banda de detalii a raportului „Situaţia mijloacelor fixe..." conţine patru câmpuri. Cel pentru codul mijlocului fix are ca sursă de date un câmp de tabelă, `mijfix.cod`.

Un efect deosebit ce poate fi folosit într-un raport este afişarea alternativă a două valori. Cu alte cuvinte, în aceeaşi poziţie a raportului va fi afişată o valoare, dacă este îndeplinită o anumită condiţie, şi alta, dacă nu este îndeplinită condiţia respectivă. De exemplu, pentru a afişa valoarea unui câmp de tip logic al unei tabele, se poate folosi în raport un câmp cu următoarea expresie ca sursă de date:

```
IIF(<câmp tabelă>,"Da","Nu")
```

Formatul de afişare al câmpului reprezintă un şir de caractere în care sunt codificate o serie de proprietăţi de afişare a valorii câmpului respectiv. Pentru detalii privind modul de alcătuire a acestui cod se poate consulta capitolul referitor la Constructorul de ecrane, paragraful care tratează câmpurile de editare ale formelor. Formatul de afişare pentru un câmp dintr-un raport se introduce fie manual în câmpul de editare **Format** (format) al ferestrei **Report Expression**, fie în mod interactiv, în fereastra deschisă la acţionarea butonului din dreapta câmpului de editare **Format**:



Tipul de date al câmpului raportului se alege din cele trei butoane alternative, **Character** (şir de caractere), **Numeric** (numeric) şi **Date** (dată calendaristică). Selectarea unuia dintre aceste butoane determină afişarea în secţiunea **Editing options** (opţiuni de editare) a diferitelor proprietăţi de afişare specifice fiecărui tip de date.

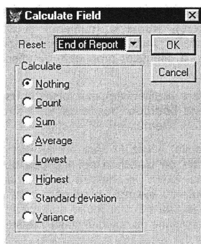
Modul de determinare a poziţiei câmpului relativ la banda în care este definit se precizează în secţiunea **Field position** (poziţie câmp) a ferestrei **Report Expression**. Avem la dispoziţie următoarele alternative:

- **Float** (poziţie mobilă) – caz în care poziţia câmpului relativ la bandă este variabilă, în funcţie de obiectele definite deasupra lui. De exemplu, dacă deasupra câmpului avem un obiect a cărei dimensiune verticală variază în funcţie de conţinut (de exemplu o adresă, care poate fi scurtă sau lungă, întinsă pe un rând sau pe mai multe etc), câmpul va fi deplasat mai jos, pentru a nu se suprapune cu obiectul plasat deasupra sa în banda respectivă;
- **Fix relative to top of band** (poziţie fixă relativ la marginea superioară a benzii) – poziţia câmpului este fixă şi se calculează relativ la marginea superioară a benzii. Dacă avem o bandă cu dimensiune variabilă (ajustabilă după conţinut – a se vedea mai jos), poziţia câmpului relativ la marginea inferioară a benzii poate să difere de la o instanţă la alta a benzii;
- **Fix relative to bottom of band** (poziţie fixă relativ la latura inferioară a benzii) – poziţia câmpului este fixă, dar ea se calculează relativ la marginea inferioară a benzii. Invers decât în cazul anterior, poziţia câmpului va fi variabilă faţă de marginea superioară, dar fixă faţă de marginea inferioară.

Uneori, conţinutul unui câmp poate varia foarte mult de la o instanţă la alta, drept pentru care o dimensiune fixă nu este adecvată: uneori câmpul va fi gol, iar alteori va fi neîncăpător pentru toate informaţiile din sursa de date. Acesta este, de exemplu, cazul unui câmp de tip memo, care prin natura sa conţine diferite texte cu dimensiuni neuniforme (o adresă, o serie de observaţii etc.). Activarea comutatorului **Stretch with overflow** (expandabil la umplere) din fereastra **Report Expression** determină extinderea în jos a câmpului raportului atât cât este necesar pentru a cuprinde tot conţinutul sursei de date respective.

Obs *De obicei, această proprietate a câmpului este combinată cu dimensiunea variabilă a benzii care conţine câmpul respectiv.*

Instanţele multiple ale benzilor unui raport determină valori multiple ale câmpurilor. Asupra acestor valori se pot aplica diferite funcţii pentru a obţine astfel informaţii sintetice. De exemplu, am putea dori însumarea tuturor valorilor unui câmp sau calculul mediei valorilor respective. Funcţia care se aplică asupra unui câmp al unui raport este precizată în fereastra de dialog **Calculate Field** (câmp calculat), deschisă la acţionarea butonului **Calculations** (calcul) a ferestrei **Report Expresion**.



Funcția se alege prin intermediul butoanelor de selecție din secțiunea **Calculate**:

- **Nothing** (nimic) – câmpul rămâne așa cum este (nu se aplică nici o funcție);
- **Count** (numărare) – în câmp se afișează numărul de valori ale sursei de date a câmpului;
- **Sum** (sumă) – se însumează valorile din sursa de date și în câmp este afișată valoarea obținută în urma însumării;
- **Average** (medie) – corespunde mediei valorilor sursei de date;
- **Lowest** (cea mai mică valoare) – în câmp va fi afișată cea mai mică valoare a sursei de date;
- **Highest** (cea mai mare valoare) – în câmp va fi afișată cea mai mare valoare a sursei de date;
- **Standard deviation** (deviația standard) – câmpul va fi completat cu deviația standard a valorilor sursei de date a câmpului respectiv;
- **Variance** (varianța) – valoarea din câmp va reprezenta varianța valorilor sursei de date.

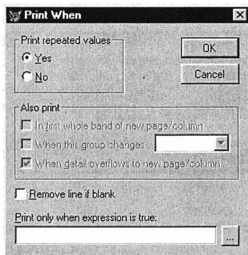
O altă posibilitate în lucrul cu câmpurile calculate prin funcțiile de mai sus este aducerea la zero a valorii câmpului cu diferite ocazii, precum începerea unei noi pagini, începerea unui nou grup (în cazul grupării datelor), terminarea unui grup (subsolul grupului etc.). Momentul anulării valorii din câmp este stabilit cu ajutorul listei derulante **Reset** (aducere la zero) a ferestrei **Calculate Field**.

Exemplu

Totalul valorii mijloacelor fixe ale categoriei se obține printr-un câmp în banda subsol de grup, câmp care are ca sursă de date câmpul `mijfix.val_invent`. *Însă asupra acestui câmp se aplică funcția de însumare (se alege butonul de selecție **Sum**), iar valoarea sa este adusă la zero la fiecare sfârșit de grup (în lista **Reset** a ferestrei **Calculate Field** se alege elementul* `mijfix.categorie`*).*

Afișarea condițională a câmpului este una dintre facilitățile deosebite. Putem să afișăm într-un raport un mesaj de avertizare care să semnaleze depășirea unei anumite valori (de exemplu, profit negativ, ceea ce înseamnă pierdere pentru o societate comercială) sau putem afișa alternativ două câmpuri, în aceeași poziție a raportului: unul dacă este îndeplinită o anumită condiție și altul dacă nu este îndeplinită condiția respectivă.

Condițiile în care este afișat un câmp sunt precizate în fereastra de dialog **Print When** (afișează când...):



deschisă la acționarea butonului cu același nume al ferestrei **Report Expression**.

O primă proprietate care poate fi stabilită prin această fereastră este afișarea valorilor repetitive. Cu alte cuvinte, în cazul a două sau mai multe valori consecutive identice ale sursei de date, se poate afișa numai prima dintre ele. Pentru aceasta, se alege butonul radio **No** din secțiunea **Print repeated values** (afișează valori repetitive).

Varianta opusă este afişarea tuturor valorilor sursei de date, indiferent dacă acestea se repetă sau nu (cazul selectării butonului de selecţie **Yes**).

Un alt caz de afişare condiţională a unui câmp este atunci când linia pe care este afişat câmpul nu conţine nimic. Dacă se activează comutatorul **Remove line if blank** (înlătură linia dacă este goală), atunci liniile complet goale (care conţin numai valori vide) nu mai sunt afişate în raport.

Condiţionarea afişării câmpului de o expresie introdusă de proiectant se face prin intermediul câmpului de editare **Print only when expression is true** (afişează numai dacă expresia este adevărată). După cum indică şi numele, în acest câmp se poate introduce o expresie care să fie evaluată înainte de afişarea câmpului în raport. Dacă expresia are valoarea logică *adevărat*, câmpul va fi afişat, iar în caz contrar, câmpul nu va apărea în raportul final.

Exemplu

De exemplu, dacă un mijloc fix se amortizează total (valoarea amortizată egalează valoarea de inventar), atunci în dreptul valorii amortizate trebuie scris mesajul „Am.total". *Pentru aceasta, în banda de detalii se definesc două obiecte suprapuse. Unul este câmpul cu valoarea de amortizare, care are drept condiţie de afişare expresia:*

```
mijfix.val_invent > mijfix.val_am
```

Cel de-al doilea obiect suprapus este un text informativ (mesajul „Am.total"*), care are de asemenea asociată o condiţie de afişare, negarea condiţiei din cazul câmpului:*

```
mijfix.val_invent <= mijfix.val_am
```

La afişare sunt evaluate cele două expresii şi se afişează obiectul care corespunde valorii adevărat.

Alte evenimente care pot condiţiona afişarea sau ascunderea unui câmp al unui raport sunt:

- apariţia primei benzi complete a unei pagini sau coloane (când se activează comutatorul **In first whole band of page/column**);
- la schimbarea grupului specificat (la activarea comutatorului **When this group changes**);
- când detaliile se întind peste o pagină sau o coloană nouă (la activarea comutatorului **When detail overflows to new page/column**).

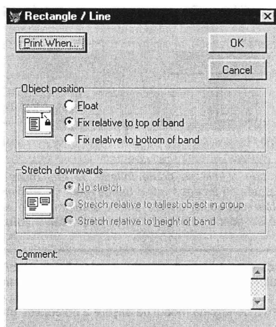
Elemente semigrafice

Am inclus în această categorie a obiectelor care pot face parte dintr-un raport liniile şi chenarele drepte sau cu colţuri rotunjite. Aceste elemente pot contribui semnificativ la aspectul plăcut şi lizibil al unui raport, la punerea în evidenţă a anumitor zone mai importante, la delimitarea diferitelor zone ale raportului.

Introducerea unei linii într-un raport se face prin acţionarea butonului al barei utilitare a obiectelor Constructorului de rapoarte, urmată de trasarea începutului şi sfârşitului liniei respective. Pentru chenarele drepte butonul este , iar pentru cele cu colţuri rotunjite .

O dată trasat elementul semigrafic dorit se poate trece la configurarea sa, adică la specificarea anumitor caracteristici.

Proprietăţile obiectului semigrafic (linie sau chenar) se specifică în fereastra deschisă ca urmare a executării unui clic dublu pe obiectul respectiv.

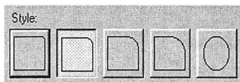


Butonul **Print When** şi fereastra deschisă la acţionarea sa se folosesc pentru precizarea condiţiilor în care va fi afişat elementul respectiv (ca şi în cazul câmpurilor). La fel se întâmplă şi cu butoanele din secţiunea **Object position**, prin intermediul cărora se indică modul în care este stabilită poziţia obiectului respectiv.

Secţiunea **Stretch downwards** este folosită pentru specificarea proprietăţilor chenarelor de a-şi adapta dimensiunea verticală în cazul în care obiectul încadrează alte obiecte (câmpuri) de dimensiune variabilă. Avem la dispoziţie următoarele alternative:

- **No stretch** (fără ajustare) – obiectul îşi păstrează dimensiunile indiferent de conţinutul său (de obiectele încadrate);
- **Stretch relative to tallest object in group** (ajustare în funcţie de cel mai înalt obiect al grupului) – chenarul va fi ajustat în funcţie de cel mai înalt obiect al grupului de obiecte pe care le încadrează;
- **Stretch relative to height of band** (ajustare în funcţie de înălţimea benzii) – dimensiunea chenarului va fi dată de înălţimea benzii în care acesta este definit. Cu alte cuvinte, distanţa de la latura inferioară a chenarului la latura inferioară a benzii va fi aceeaşi, indiferent de înălţimea benzii respective; la fel şi pentru latura superioară.

În cazul chenarelor cu colţuri rotunjite, fereastra de dialog conţine o secţiune referitoare la curbura colţurilor:



Obs *Un chenar sau o linie se poate întinde pe mai multe benzi. Un obiect semigrafic poate să înceapă într-o bandă şi să se termine în alta. O bandă poate fi chiar petrecută de un obiect semigrafic. În acest cazuri, este necesar să se controleze foarte strict proprietăţile obiectelor semigrafice respective.*

Stilul liniei cu care se trasează obiectul semigrafic se stabileşte prin submeniul **Pen** (peniţă) al meniului **Format**. Textura de umplere (în cazul chenarelor) este dată de opţiunea aleasă din submeniul **Fill** (umplere) al meniului **Format**, iar submeniul **Mode** (mod) al aceluiaşi meniu determină modul de afişare a obiectului, transparent sau opac.

Imagini

Un alt tip de obiecte ce se pot defini în cadrul benzilor unui raport sunt imaginile. Acestea trebuie să fi fost create anterior prin intermediul unui program grafic, urmând ca în raport să fie preluat fişierul (de tip grafic) în care a fost depozitată imaginea respectivă.

Exemplu

De exemplu, în raportul referitor la mijloacele fixe ale unităţii economice, sigla firmei este o imagine construită cu un program de editare grafic. În raport, ea a fost importată cu ajutorul unui obiect de tip imagine.

Crearea unei imagini într-o bandă a raportului se face cu ajutorul butonului de pe bara utilitară a obiectelor Constructorului de rapoarte.

Obs

Pentru introducerea unei imagini într-un raport se foloseşte tehnologia OLE de încorporare şi legare a obiectelor (a se vedea paragraful dedicat acestui subiect).

După acţionarea butonului pentru imagini, se trasează zona din raport în care va fi plasată imaginea respectivă; în urma acestei operaţii, se deschide o fereastră de dialog pentru specificarea parametrilor imaginii. În această fereastră se indică fişierul (câmpul de editare **File**) sau câmpul tabelei (câmpul de editare **Field**) din care va fi preluată imaginea, condiţiile de afişare a imaginii (butonul **Print When**) şi modul de afişare (secţiunea **If picture and frame are different sizes** – dacă imaginea şi zona din raport au dimensiuni diferite). În ceea ce priveşte această ultimă proprietate, proiectantul raportului are următoarele alternative:

- **Clip picture** – în acest caz, imaginea îşi păstrează dimensiunile originale şi din ea se afişează numai porţiunea care încape în cadru;
- **Scale picture, retain shape** – se redimensionează imaginea păstrându-se proporţiile originale, astfel încât imaginea să încapă cât mai bine în cadru;
- **Scale picture, fill the frame** – se redimensionează imaginea, dar nu mai sunt păstrate proporţiile originale, imaginea fiind eventual deformată pentru a umple complet cadrul rezervat în raport.

Exemplu

*Imaginea din raportul mijloacelor fixe a fost importată din fişierul `SIGLA.BMP` (specificat în câmpul de editare **File** al ferestrei de proprietăţi a figurii). Modul de adaptare a dimensiunilor originale ale figurii la cele ale zonei rezervate în raport este **Scale picture, fill the frame**, deoarece proporţiile imaginii nu sunt esenţiale, în schimb, dimensiunile zonei rezervate în raport nu trebuie modificate.*

Gruparea datelor în raport. Niveluri de grupare

Gruparea datelor reprezintă una dintre facilităţile oferite de Constructorul de rapoarte. Cu alte cuvinte, cu ajutorul acestui utilitar se pot construi rapoarte cu unul sau mai multe niveluri de grupare, sau de totalizare. Fiecare criteriu de grupare este caracterizat printr-o expresie, care se evaluează pentru fiecare instanţă a benzii. Vor forma un grup acele instanţe care corespund aceleiaşi valori a expresiei de grupare. Prin urmare, într-un raport cu un nivel de grupare vom avea atâtea grupuri câte valori are cheia de grupare.

Pentru construirea unui raport cu unul sau mai multe niveluri de grupare, este necesară specificarea criteriilor de grupare şi a conţinutului benzilor de antet şi de subsol pentru fiecare nivel. Un nivel de grupare adăugat la raport aduce cu sine o bandă de antet de grup (afişată la începutul fiecărui grup de nivelul respectiv) şi una de subsol de grup (afişată după fiecare grup de nivelul respectiv).

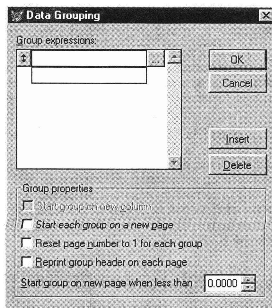
Exemplu

De exemplu, raportul „Situaţia mijloacelor fixe ..." *conţine un singur nivel de grupare, cel dat de categoria mijlocului fix. Expresia de grupare este deci:*

```
mijfix.categorie
```

Specificarea expresiilor după care se construiesc nivelurile de grupare se face în fereastra **Data Grouping** (gruparea datelor), deschisă la alegerea opţiunii cu acelaşi nume a meniului **Report** (a se vedea figura următoare).

Lista **Group expressions** (expresii de grupare) a acestei ferestre conţine câte o linie pentru fiecare nivel de grupare. Fiecare linie conţine un câmp de editare în care se poate specifica direct expresia de grupare a nivelului respectiv. Dacă pentru construirea acestei expresii se doreşte folosirea Constructorului de expresii, se execută clic pe butonul din dreapta câmpului de editare.



Butonul din stânga câmpului se foloseşte la interschimbarea criteriilor de grupare, dat fiind faptul că poziţia în listă este esenţială. Prima linie corespunde nivelului unu de grupare, cel mai cuprinzător, a doua linie corespunde nivelului doi de grupare, cel care realizează gruparea în interiorul grupurilor de nivel unu şi aşa mai departe. Mutarea unui criteriu de grupare de pe o poziţie din listă pe alta se face prin tragerea cu mouse-ul a butonului din stânga listei.

Inserarea unui nou criteriu de grupare în poziţia curentă a cursorului se face prin acţionarea butonului **Insert** (inserare), iar ştergerea criteriului pe care se află cursorul este posibilă cu ajutorul butonului **Delete** (ştergere).

Fiecare criteriu de grupare are asociate o serie de proprietăţi, care pot fi configurate prin obiectele de interfaţă din secţiunea **Group properties** (proprietăţile grupului). În cazul activării comutatorului **Start group on new column** (începe grupul în coloană nouă), fiecare grup al nivelului respectiv începe pe o nouă coloană (bineînţeles, în cazul unui raport multicoloană). Cu alte cuvinte, dacă un grup se termină la mijlocul unei coloane, restul spaţiului din coloana respectivă este lăsat liber şi următorul grup începe pe coloana următoare.

La fel se foloseşte şi comutatorul **Start each group on a new page** (începe fiecare grup pe o pagină nouă), care determină afişarea fiecărui grup pe o pagină nouă.

Numerotarea paginilor raportului poate fi, de asemenea, influenţată de gruparea datelor. După cum îi spune şi numele, comutatolul **Reset page number to 1 on each**

group (renumerotarea paginilor de la 1 pentru fiecare grup) determină începerea fiecărui grup pe pagină nouă şi, în plus, renumerotarea paginilor raportului (de la 1) pentru fiecare grup.

În fine, activarea comutatorului **Reprint group header on each page** (tipăreşte antetul grupului pe fiecare pagină) determină tipărirea benzii antet de grup la începutul fiecărei pagini a raportului (chiar dacă acolo nu începe un grup nou), imediat după antetul de pagină.

Începerea unui grup nou la sfârşitul unei pagini nu este totdeauna indicată; uneori, este preferabilă trecerea la pagină nouă, de exemplu atunci când pe pagina curentă a rămas un spaţiu mai mic decât cel prestabilit. Distanţa maximă de la marginea inferioară a paginii până la începutul antetului de grup pentru care se permite începerea unui grup nou se specifică în câmpul de editare **Start group on new page when less than...** (începe grupul pe pagină nouă când spaţiul este mai mic decât ...). În cazul în care spaţiul rămas liber pe pagina curentă este mai mic decât valoarea specificată, noul grup va începe pe o pagină nouă.

Folosirea variabilelor în construirea rapoartelor

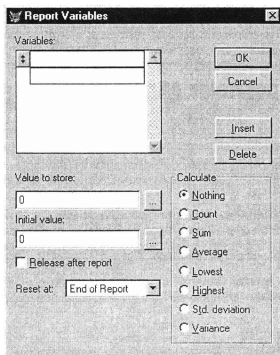
În cadrul unui raport este necesară deseori folosirea unor variabile care să memoreze diferite valori temporare, utile la efectuarea unor calcule.

De exemplu, dacă dorim efectuarea unei sume pentru mai multe înregistrări, sumă care ulterior va fi folosită în mai multe locuri ale raportului, este de preferat folosirea unei variabile în care să se depună rezultatele calculelor respective şi apoi plasarea variabilei în locurile corespunzătoare, decât introducerea formulei de calcul în fiecare loc în care este nevoie de valoarea sumei (pentru a se evita recalcularea).

Definirea variabilelor unui raport se face în fereastra **Report Variables** (variabile raport), deschisă la alegerea opţiunii **Variables** a meniului **Report**.

Numele fiecărei variabile va fi introdus în lista **Variables** (variabile). Ordinea în listă este semnificativă, determinând ordinea de calculare a variabilelor respective. O dată selectată o variabilă din listă, se pot specifica pentru ea o serie de caracteristici, printre care:

- modul de calcul al variabilei (expresia după care se calculează valoarea sa) în câmpul **Value to store** (valoarea de memorat);
- valoarea iniţială a variabilei, în câmpul de editare **Initial value**;



- momentul în care variabila este readusă la valoarea inițială, moment specificat cu ajutorul listei **Reset at**. Avem la dispoziție variante precum *End of Report* (sfârşit de raport), *End of Page* (sfârşit de pagină), *End of Column* (sfârşit de coloană) etc. Cu alte cuvinte, apariția evenimentului respectiv determină readucerea la valoarea inițială a variabilei şi continuarea calculului de la valoarea respectivă;
- comutatorul **Release after report** (ştergere după raport) face ca variabila să fie eliminată din memorie la terminarea rulării raportului;
- secțiunea **Calculate** (calcul) conține butoane alternative care corespund diferitelor funcții ce pot fi aplicate pentru calcularea valorii variabilei respective. De exemplu, alegându-se butonul **Sum** (sumă), valoarea variabilei se va calcula ca sumă a valorilor expresiei specificate pentru fiecare instanță a benzii.

Exemplu

În raportul folosit ca exemplu în acest capitol se foloseşte o variabilă predefinită pentru afişarea în josul paginii a numărului acesteia. Variabila este `_pageno`.

> *Deşi totalizarea valorilor mijloacelor fixe s-a făcut prin intermediul unor câmpuri asupra cărora s-a aplicat funcţia de însumare, s-ar fi putut folosi ca sursă de date şi variabile, care ar fi fost calculate tot prin însumare.*

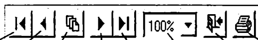
Rularea rapoartelor. Afişarea pe ecran şi tipărirea

După proiectarea unui raport cu ajutorul Constructorului de rapoarte şi salvarea sa, fişierul creat poate fi folosit pentru construirea efectivă a raportului şi afişarea lui pe ecran sau tipărirea pe hârtie. Această fază este „rularea raportului". Un modul special al sistemului citeşte datele din fişierul raport şi datele din tabelele sursă şi construieşte astfel raportul propriu-zis.

Comanda folosită pentru apelarea modulului de rulare a rapoartelor este REPORT FORM. Pentru afişarea pe ecran a unui raport se foloseşte următoarea construcţie:

```
REPORT FORM <raport> PREVIEW
```

Ca urmare a acestei comenzi, raportul este construit şi afişat pe ecran într-o fereastră, iar la fereastra sistemului Visual FoxPro este adăugată o bară utilitară ca aceea de mai jos:



Prin urmare, din această fereastră se poate comanda şi tipărirea raportului la imprimantă, prin intermediul butonului .

Rularea raportului şi trimiterea sa direct la imprimantă (fără previzualizare) se face cu ajutorul construcţiei:

```
REPORT FORM <raport> NOCONSOLE TO PRINTER PROMPT
```

Clauza NOCONSOLE se foloseşte pentru inhibarea afişării raportului pe ecran, iar clauza PROMPT este opţională; în prezenţa acesteia, înainte de tipărire este afişată fereastra de specificare a parametrilor de tipărire.

Trimiterea raportului spre un fişier în loc de imprimantă se realizează cu ajutorul clauzei TO FILE a comenzii REPORT FORM:

```
REPORT FORM <raport> TO FILE <fişier> NOCONSOLE ASCII
```

Clauza NOCONSOLE se foloseşte atunci când nu se doreşte ecou pe ecran în timpul afişării în fişier. Clauza ASCII determină modul de scriere în fişier: în prezenţa clauzei formatul fişierului va fi text, sau ASCII, iar în absenţa clauzei formatul va fi unul specific imprimantei (informaţiile se depozitează în fişierul respectiv, exact cum s-ar trimite la imprimantă).

Obs *De obicei, comanda de rulare a raportului se introduce în metoda **Click** a butonului **Raport** din forma programului de raportare.*

Meniuri. Constructorul de meniuri

Capitolul 11

- Introducere
- *Constructorul de meniuri – mod de folosire*
  - ✓ Pornirea Constructorului de meniuri
  - ✓ Proprietăți globale ale meniului
  - ✓ Fereastra de lucru a Constructorului de meniuri
  - ✓ Proprietățile submeniurilor
  - ✓ Caracteristicile opțiunilor meniurilor
- Generarea programului de construire a meniului. Rularea și activarea meniului

Introducere

Ce este un meniu şi la ce foloseşte?

Interfeţele cu utilizatorul ale sistemelor informatice conţin mai multe tipuri de elemente, precum ferestre de dialog, obiecte de interfaţă, indicatoare grafice etc.; printre acestea se numără şi meniurile. Aproape fiecare sistem informatic conţine, într-o formă sau alta, un meniu.

Def

***Meniul** reprezintă un ansamblu de opţiuni puse la dispoziţia utilizatorului pe ecranul monitorului, opţiuni la alegerea cărora sunt declanşate diferite operaţii de prelucrare.*

De obicei, meniul apare în partea superioară a ferestrei unei aplicaţii şi are o structură standard. În cadrul unui meniu sunt incluse diferite opţiuni, la alegerea cărora sunt declanşate prelucrările ce pot fi realizate în aplicaţia respectivă. În general, orice operaţie care poate fi executată în cadrul unei aplicaţii trebuie să aibă corespondent în meniul acesteia, pentru a putea fi astfel declanşată. Meniul reprezintă deci mijlocul prin care utilizatorul comunică sistemului ce operaţie să efectueze la un moment dat.

Meniul se foloseşte în aproape toate aplicaţiile care depăşesc un nivel elementar de complexitate, adică în care există mai multe operaţii de executat şi deci mai multe alternative la dispoziţia utilizatorului.

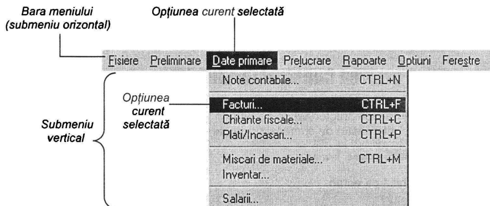
Structura standard a unui meniu. Elemente componente

Deşi meniurile pot fi de mai multe tipuri (grafice sau în modul text, orizontale, verticale sau neregulate etc.), de-a lungul evoluţiei interfeţelor cu utilizatorul s-a conturat o structură standard a acestor tipuri de elemente. Un meniu este format, în general, dintr-o bară (sau un submeniu orizontal) care conţine mai multe opţiuni. Fiecare dintre acestea are asociat un submeniu vertical, care este activat numai la alegerea opţiunii respective. Alegerea de către utilizator a opţiunilor meniurilor verticale poate declanşa o operaţie de prelucrare sau poate determina afişarea unui nou submeniu vertical.

Prin urmare, un meniu este compus din **submeniuri** (orizontale sau verticale). Un submeniu are în componenţă **opţiuni**, care pot fi şi ele de două tipuri: declanşatoare de operaţii de prelucrare sau căi spre alte submeniuri.

Obs

În continuare, atunci când este exclusă confuzia, vom desemna prin termenul „meniu" şi submeniurile componente ale unui meniu complex.



Proiectarea unui meniu constă în specificarea elementelor componente (submeniuri şi opţiuni) şi a caracteristicilor acestora (legate de aspect şi de comportamentul în diferite situaţii).

Cum se construieşte şi cum se foloseşte un meniu?

Ca şi versiunile anterioare, limbajul Visual FoxPro are implementate comenzi şi funcţii pentru definirea şi manipularea meniurilor. Prin urmare, putem construi un meniu prin intermediul unui program care conţine astfel de comenzi, aceasta fiind metoda clasică de construire a meniurilor.

Visual FoxPro (dar şi FoxPro 2.5 sau 2.6) posedă instrumente interactive de construire a meniurilor, principalul fiind Constructorul de meniuri. Acesta este un utilitar al mediului care preia în mod interactiv de la utilizator preferinţele lui în ceea ce priveşte meniul şi, pe baza acestora, generează automat programul care construieşte meniul respectiv. Această metodă este recomandată pentru construirea meniurilor aplicaţiilor Visual FoxPro, deoarece este mai rapidă şi mai la îndemână decât cea clasică.

Constructorul de meniuri realizează colectarea interactivă, de la utilizator, a parametrilor de construire a meniului şi depozitarea lor într-un fişier special. Acest fişier reprezintă de fapt o tabelă cu o structură specială (cu extensia implicită .MNX), construită şi interpretată de către Constructor. Prin urmare, prima etapă a proiectării unui meniu constă în specificarea opţiunilor şi depunerea lor în tabela meniului.

Urmează o nouă etapă, aceea a generării, pe baza specificaţiilor proiectantului, a programului care construieşte efectiv meniul. O dată generat, programul poate fi rulat printr-o simplă comandă DO. Comanda de rulare poate fi inclusă (şi de obicei este) în programul monitor al sistemului informatic.

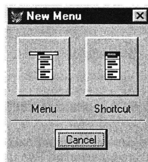
Constructorul de meniuri – mod de folosire

Pornirea Constructorului de meniuri

Pentru pornirea Constructorului de meniuri în vederea proiectării unui meniu nou, se poate introduce în fereastra de comenzi următoarea instrucţiune:

```
CREATE MENU <nume meniu>
```

Din fereastra deschisă pe ecran:



se alege butonul **Menu** (meniu), dacă se doreşte crearea unui meniu cu bară orizontală şi submeniuri verticale, respectiv butonul **Shortcut** (scurtătură), dacă se doreşte construirea unui submeniu vertical, având subordonate alte meniuri verticale. Acţionarea butonului corespunzător va determina deschiderea ferestrei de lucru a Constructorului de meniuri.

Varianta interactivă de pornire a Constructorului de meniuri constă în alegerea opţiunii **New** a meniului **File**, urmată de selectarea butonului de selecţie **Menu** (meniu) al ferestrei de dialog deschise pe ecran şi acţionarea butonului **New file** al aceleiaşi ferestre.

Modificarea unui meniu creat anterior se face cu ajutorul instrucţiunii:

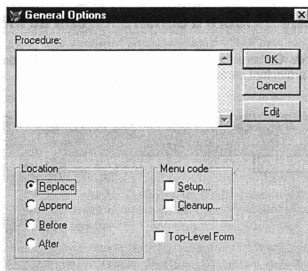
```
MODIFY MENU <nume meniu>
```

care se introduce în fereastra de comenzi sau prin alegerea opţiunii **Open** a meniului **File** şi urmată de alegerea meniului de modificat într-o fereastră de dialog.

Proprietăţi globale ale meniului

Înainte de specificarea structurii unui meniu (opţiunile componente şi caracteristicile acestora, gruparea lor în submeniuri etc.), trebuie stabilite o serie de caracteristici ale meniului în ansamblul său, cum ar fi poziţia faţă de meniul sistemului Visual FoxPro.

Proprietăţile globale ale meniului sunt stabilite în fereastra de dialog **General Options** (opţiuni generale), deschisă la alegerea opţiunii cu acelaşi nume a submeniului **View**:



Poziţia noului meniu relativ la meniul standard al sistemului este stabilită prin intermediul butoanelor din secţiunea **Location** (locaţie). Noul meniu va înlocui meniul sistem, atunci când se alege butonul de selecţie **Replace** (înlocuire), sau va fi adăugat la meniul sistem, atunci când butonul radio ales este **Append** (adăugare). Alegerea butonului **Before** (înainte) determină plasarea noului meniu înaintea submeniului sistemului specificat în lista derulantă alăturată (care este afişată numai la selectarea butonului). La fel funcţionează şi butonul **After** (după), numai că noul meniu va fi plasat după submeniul specificat.

Pentru un meniu, există posibilitatea specificării unor secvenţe de comenzi care să fie executate cu diferite ocazii (dirijarea prin evenimente). Astfel, la activarea meniului poate fi executată o procedură al cărei cod este introdus de proiectant în zona **Procedure** (procedură) sau în fereastra deschisă la acţionarea butonului **Edit**.

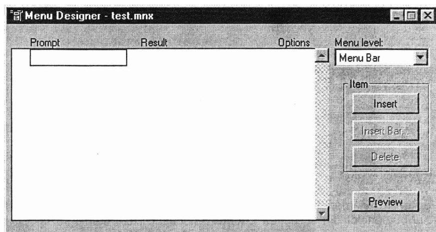
Alte evenimente pentru care se pot specifica secvenţe de cod ale proiectantului sunt iniţializarea meniului (crearea sa) şi ştergerea meniului (eliminarea sa din memorie). Secvenţele de cod respective se specifică în ferestre deschise prin activarea comutatoarelor **Setup** (iniţializare) şi **Cleanup** (ştergere), urmată de acţionarea butonului **OK**.

Exemplu

*De exemplu, crearea unui meniu ar putea implica citirea unor date dintr-un fişier, iar comenzile de citire respective trebuie introduse în secvenţa de cod ataşată evenimentului **Setup**.*

Fereastra de lucru a Constructorului de meniuri

După pornirea Constructorului, pe ecran este deschisă fereastra în care sunt precizate de către proiectant caracteristicile meniului:



Fereastra conţine o listă în care sunt enumerate şi descrise opţiunile meniului creat şi o serie de obiecte de interfaţă folosite la manipularea acestor opţiuni. La un moment dat, în fereastra de lucru este afişat numai grupul de opţiuni al unui submeniu (orizontal sau vertical) al meniului proiectat.

O opţiune poate fi folosită pentru accesul la alt submeniu, conţinând propriul grup de opţiuni. Acţionarea butonului din coloana a patra a listei opţiunilor (**Create** sau **Edit**) determină intrarea în editarea submeniului subordonat opţiunii respective, adică

încărcarea opţiunilor submeniului respectiv în lista ferestrei. Pentru a reveni la meniul imediat superior se foloseşte lista derulantă **Menu level** (nivel meniu), în care sunt disponibile toate meniurile superioare ierarhic meniului curent.

În lista ferestrei, pe verticală, sunt specificate opţiunile meniului, fiecare pe câte o linie a listei. Coloanele listei sunt folosite pentru descrierea fiecărei opţiuni în parte, astfel:

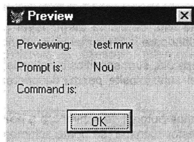
- prima coloană conţine butonul de schimbare a poziţiei opţiunii – prin tragerea cu mouse-ul a acestor butoane, se poate schimba poziţia unei opţiuni relativ la celelalte;
- în coloana a doua (**Prompt**) se introduce textul explicativ al opţiunii – textul care este afişat în meniu, pe poziţia opţiunii respective;
- coloana a treia (**Result** – rezultat) este rezervată rezultatului selectării opţiunii şi cuprinde o listă derulantă prin intermediul căreia este stabilit efectul obţinut prin alegerea opţiunii respective (activarea unui submeniu, executarea unei comenzi etc.);
- cea de-a patra coloană se foloseşte pentru specificarea operaţiei de executat la alegerea opţiunii. Dacă opţiunea are asociat un submeniu, atunci în această coloană se va afla un buton prin acţionarea căruia se intră în editarea submeniului respectiv. Dacă opţiunea are asociată o comandă, în coloana aceasta va fi prezent un câmp de editare în care proiectantul va preciza comanda de executat;
- coloana a cincea, numită **Options** (opţiuni), conţine un buton la a cărui acţionare este deschisă o fereastră de dialog folosită pentru parametrizarea comportamentului opţiunii.

Butoanele din secţiunea **Item** (articol) sunt folosite pentru adăugarea de noi elemente între cele existente (în meniul curent) – butoanele **Insert** şi **Insert Bar** – şi pentru ştergerea unor opţiuni existente – butonul **Delete**.

Previzualizarea meniului

Previzualizarea meniului înseamnă afişarea de probă a meniului editat în fereastra de lucru a Constructorului, chiar dacă el nu a fost salvat sau programul corespunzător nu a fost încă generat. La previzualizare, alegerea unei opţiuni a meniului nu este însoţită de execuţia comenzii corespunzătoare, ci doar de afişarea acesteia.

Previzualizarea meniului în curs de editare se realizează prin acţionarea butonului **Preview** (previzualizare) din fereastra Constructorului de meniuri sau prin alegerea opţiunii **Preview** a submeniului **Menu**. În starea de previzualizare, meniul sistemului este înlocuit de meniul în curs de editare, iar pe ecran este afişată o fereastră care indică opţiunea selectată din acest meniu:

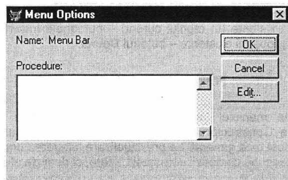


Ieşirea din starea de previzualizare se face prin acţionarea butonului **OK**.

Proprietăţile submeniurilor

Un meniu complex este alcătuit din mai multe submeniuri, orizontale sau verticale. Pentru fiecare submeniu se poate specifica o secvenţă de comenzi care să fie rulată la alegerea unei opţiuni a submeniului respectiv. Comenzile acestea nu vor fi executate atunci când se alege o opţiune a submeniului care are specificată explicit o operaţie de executat, ci numai în cazul opţiunilor pentru care nu s-a specificat explicit ce se va întâmpla la alegerea lor.

Pentru specificarea acestui cod, este necesar ca, în momentul respectiv, în fereastra de lucru a Constructorului de meniuri să se afle în editare submeniul dorit. În aceste condiţii se alege opţiunea **Menu Options** (opţiuni meniu). În fereastra deschisă pe ecran:



în secţiunea **Procedure** (procedură) sau în fereastra deschisă la acţionarea butonului **Edit** (editare), se poate specifica secvenţa de cod dorită.

Caracteristicile opțiunilor meniului

O dată specificate caracteristicile de ansamblu ale meniului, se poate trece la precizarea elementelor sale, adică a opțiunilor și a proprietăților acestora. O opțiune a meniului editat cu ajutorul Constructorului de meniuri se caracterizează în primul rând prin textul care este afișat în meniu pe poziția respectivă (textul informativ al opțiunii) și prin efectul produs la acționarea sa. Aceste două proprietăți sunt obligatorii pentru orice opțiune de meniu.

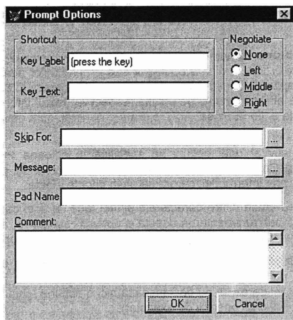
Textul informativ al unei opțiuni se specifică în coloana **Prompt** a listei din fereastra de lucru a Constructorului. În acest șir de caractere se pot introduce niște combinații speciale, prin intermediul cărora se obțin diferite efecte:

- \< înaintea unuia dintre caracterele șirului face ca acel caracter să fie folosit ca o tastă pentru selectarea directă a opțiunii respective, atunci când meniul este activat. Caracterul respectiv apare subliniat în textul opțiunii;
- \ înaintea textului opțiunii face ca opțiunea respectivă să fie dezactivată, adică să nu fie accesibilă utilizatorului. Aceasta apare pe ecran în culori șterse, iar alegerea sa nu declanșează nici o acțiune (chiar dacă în cazul său au fost specificate operații de executat);
- \- în locul textului informativ face ca linia respectivă din meniu să nu fie considerată o opțiune, ci o bară delimitatoare între mai multe grupuri de opțiuni ale aceluiași submeniu.

Operațiile care pot fi executate la alegerea unei opțiuni sunt următoarele:

- activarea unui nou submeniu. Pentru aceasta, din lista derulantă din coloana **Result** se alege elementul *Submenu*. Coloana următoare va conține un buton la a cărei acționare se va intra în editarea submeniului respectiv;
- executarea unei anumite comenzi FoxPro. Alegerea din lista derulantă a elementului *Command* face ca opțiunea să lanseze comanda specificată în câmpul de editare din dreapta listei (coloana a patra);
- executarea mai multor comenzi grupate într-o procedură. Elementul listei derulante **Result** care corespunde acestei variante este *Procedure*. Butonul din dreapta listei permite intrarea în editarea procedurii respective.

Ultima coloană a listei, numită **Options** (opțiuni), conține un buton la a cărui acționare se deschide fereastra de dialog **Prompt Options**, folosită la specificarea unor parametri suplimentari ai opțiunii respective.



O opțiune poate avea o cale directă de alegere. Aceasta reprezintă o combinație de taste prin care se obține același efect cu alegerea opțiunii. Combinația respectivă de taste se introduce în câmpul de editare **Key Label** (etichetă tastă) din secțiunea **Shortcut** (scurtătură) – de fapt, se apasă combinația de taste dorită, câmpul fiind completat automat cu un cod al combinației respective.

Textul suplimentar care va fi atașat textului opțiunii (pentru a indica faptul că opțiunea poate fi aleasă direct prin combinația respectivă de taste) este precizat în câmpul **Key Text** (text tastă).

Obs *Atunci când numărul opțiunilor unui meniu este mare, se vor menționa căi directe de selectare numai pentru opțiunile folosite cel mai frecvent. Secvențele respective de comenzi trebuie să fie cât mai sugestive. De exemplu, se poate alege Ctrl+S pentru Salvare, Ctrl+D pentru Deschidere, Ctrl+R pentru Rulare etc.*

Accesul la o opțiune poate fi condiționat de îndeplinirea unei condiții impuse de proiectant. De exemplu, dacă nu este deschisă nici o tabelă, atunci opțiunea de editare a tabelei curente nu are sens și ea trebuie dezactivată. Impunerea unei condiții care să controleze accesul la o opțiune se face cu ajutorul câmpului de editare **Skip For** (sărită când...) al ferestrei **Prompt Options**.

În acest câmp este introdusă o expresie logică. Dacă valoarea ei este *adevărat*, atunci opţiunea va fi dezactivată, deci nu va fi disponibilă pentru utilizator (ea apare cu culori şterse, pentru a indica această stare). Dacă valoarea expresiei este *fals*, opţiunea va putea fi aleasă de utilizator.

Exemplu

Pentru ca o opţiune să fie accesibilă numai dacă un anumit fişier există pe disc (să zicem fişierul `alfa.dat`*), condiţia introdusă în câmpul **Skip For** ar putea fi:*

```
NOT(FILE("alfa.dat"))
```

Se observă aplicarea operatorului `NOT`*, datorită faptului că funcţia* `FILE()` *returnează* adevărat *dacă fişierul există şi* fals *în caz contrar. Pentru opţiune, valoarea* adevărat *indică indisponibilitatea, iar valoarea* fals *disponibilitatea.*

*Dacă în câmpul **Skip For** se introduce* `.T.`*, atunci opţiunea va fi dezactivată tot timpul. Această metodă este echivalentă cu plasarea caracterului / în faţa textului informativ al opţiunii (se obţine acelaşi efect).*

Obs

Ori de câte ori este posibil, este indicat a fi dezactivate opţiunile care nu au sens la un moment dat. În acest fel, sistemul informatic este protejat împotriva greşelilor utilizatorilor, evitându-se o serie de complicaţii ulterioare. Este mai bine să prevenim greşelile decât să le corectăm ulterior.

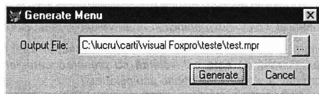
Atunci când opţiunea este selectată, poate fi afişat un mesaj explicativ în bara de stare a sistemului (plasată în partea inferioară a ferestrei Visual FoxPro). Acest mesaj se precizează de către proiectant în câmpul de editare **Message** (mesaj) al ferestrei **Prompt Options**.

Obs

Afişarea mesajelor informative asociate opţiunilor este benefică, oferind un ajutor contextual foarte util. De cele mai multe ori, în mesajul din bara de stare este explicitat textul informativ al opţiunii, adică este explicată pe scurt operaţia care va fi executată la alegerea opţiunii respective. Deci folosiţi din plin această facilitate.

Generarea programului de construire a meniului. Rularea şi activarea meniului

O dată proiectat meniul dorit în fereastra Constructorului de meniuri, el este salvat într-un fişier special (o tabelă cu extensia .MNX). Pentru a putea folosi meniul, trebuie generat un program la a cărui rulare meniul să fie activat. Această operaţie este executată prin alegerea opţiunii **Generate** (generare) a submeniului Menu. În fereastra deschisă pe ecran:



se precizează numele fişierului ce va fi generat. În mod implicit, acesta are acelaşi nume cu cel al tabelei în care este memorat meniul, dar extensia .MPR.

O dată generat meniul, acesta este efectiv afişat pe ecran (activat) la execuţia programului generat, declanşată printr-o comandă DO:

```
DO <meniu>.MPR
```

Deoarece comanda DO presupune extensia implicită .PRG (de la programe), este necesară specificarea extensiei .MPR (sau .MPX, de la varianta compilată a programului).

Asamblarea componentelor într-un sistem informatic

Capitolul 12

- Programe monitor
  - ✓ Ce este şi ce trebuie să asigure un program monitor?
  - ✓ Structura unui program monitor
- Proiecte. Gestionarea componentelor unui sistem informatic
- Generarea aplicaţiilor şi a programelor executabile
- Distribuirea sistemelor informatice

Programe monitor

Ce este şi ce trebuie să asigure un program monitor?

Un sistem informatic este compus din mai multe programe, de diferite tipuri (de introducere a datelor, de prelucrare, de raportare etc.), care conlucrează pentru executarea sarcinilor primite de la utilizator. Pentru a putea funcţiona, aceste programe trebuie coordonate, sarcină care revine programului monitor.

Def

*Prin urmare, **programul monitor** al unui sistem informatic este acel program prin care se realizează coordonarea, sincronizarea şi colaborarea între programele componente ale sistemului.*

Dintre sarcinile care revin programului monitor menţionăm:

- configurarea mediului, adică stabilirea cadrului în care urmează să funcţioneze sistemul. Acest lucru este necesar datorită faptului că parametrii de configurare impliciţi ai mediului Visual FoxPro nu sunt întotdeauna cei mai potriviţi pentru aplicaţia în cauză şi uneori sunt necesare parametrizări specifice. De exemplu, ar putea fi necesară stabilirea formatului datelor calendaristice, a locului în care sunt depozitate datele (bazele de date) etc.;
- activarea meniului principal al sistemului. Din cadrul acestui meniu se lansează în execuţie programele sistemului, adică se afişează formele, se declanşate prelucrările, se construiesc rapoartele etc.;
- pornirea procesorului de evenimente. Aceasta este o operaţie tehnică necesară datorită mecanismului implementat în Visual FoxPro. Procesorul de evenimente este un modul al sistemului care preia evenimentele apărute în sistem (apăsarea unei taste, mişcarea mouse-ului, executarea unui clic cu mouse-ul etc.) şi le transmite spre prelucrare aplicaţiei;
- în finalul lucrului cu sistemul informatic, sunt necesare refacerea stării anterioare a mediului, pentru refolosire, precum şi o serie de operaţii finale, specifice fiecărui sistem în parte (cum ar fi, de exemplu, salvarea unor parametri în fişiere).

Să vedem în cele ce urmează cum şi unde se realizează aceste operaţii.

Structura unui program monitor

Salvarea şi refacerea mediului

Să vedem mai întâi cum sunt realizate operaţiile prezentate anterior; vom începe cu salvarea mediului curent de date. Această operaţie este necesară atunci când se lucrează în mediul SGBD nu numai cu sistemul informatic respectiv, ci şi cu alte programe (aplicaţii simple sau sisteme informatice) care, de cele mai multe ori, necesită configurări specifice.

Prin mediu de date înţelegem ansamblul de parametri care controlează funcţionarea SGBD, precum formatul datei calendaristice, modul de tratare a înregistrărilor marcate pentru ştergere, modul de deschidere a tabelelor la utilizarea în reţea, locul implicit în care sistemul caută datele etc. Majoritatea parametrilor sunt stabiliţi în Visual FoxPro prin comenzi de tip `SET`.

Exemplu

De exemplu, comanda:

```
SET TALK OFF
```

inhibă ecoul pe ecran al diferitelor comenzi de prelucrare, iar comanda:

```
SET CENTURY ON
```

face ca anul din datele calendaristice să fie specificat cu patru cifre în loc de două.

Pentru a afla starea unui parametru al mediului se foloseşte funcţia `SET()`, căreia i se transmite numele parametrului de mediu dorit şi de la care se obţine în schimb valoarea sa curentă.

Exemplu

Comanda următoare memorează în variabila `salv_TALK` *starea curentă a parametrului* `TALK` *al sistemului:*

```
salv_TALK = SET("TALK")
```

Iată câtiva dintre cei mai folosiţi parametri de tip `SET` ai mediului Visual FoxPro, care trebuie precizaţi la intrarea într-un sistem informatic:

CENTURY	Stabileşte formatul anului (cu 2 sau cu 4 cifre) în datele calendaristice.
CONFIRM	Permite sau nu ieşirea automată dintr-un câmp de editare după completarea acestuia.
CONSOLE	Activează sau inhibă afişarea în fereastra activă (Visual FoxPro sau una definită de utilizator).
DATE	Stabileşte formatul datelor calendaristice.
DEFAULT	Indică discul şi directorul curent, din care se preiau programele şi datele.
DELETED	Folosit pentru controlul accesului la înregistrările marcate pentru ştergere.
ESCAPE	Tratează evenimentul apăsării tastei Escape (care poate determina sau nu oprirea execuţiei programului curent).
EXCLUSIVE	Stabileşte modul exclusiv (monoutilizator) de deschidere a tabelelor.
SAFETY	Activează sau dezactivează afişarea mesajelor de eroare şi confirmare în diferite situaţii critice (de exemplu, la suprascrierea unui fişier).
TALK	Activează sau inhibă ecoul pe ecran al comenzilor de prelucrare.

Refacerea unui parametru salvat anterior într-o variabilă se poate realiza cu ajutorul macrosubstituţiei. Aceasta determină înlocuirea numelui unei variabile cu valoarea sa. Pentru a preciza că asupra unei variabile se aplică macrosubstituţia, numele ei este precedat de &.

Exemplu

```
SET TALK &salv_TALK
```

Comanda de mai sus reface starea parametrului TALK *din variabila* **salv_TALK** *în care a fost salvat anterior.*

Activarea meniului principal al sistemului informatic

Activarea meniului principal al sistemului informatic se face prin rularea programului generat cu ajutorul Constructorului de meniuri (varianta cu extensia .MPR). Comanda are forma:

```
DO <meniu>.MPR
```

Amănunte despre modul de construire şi activare a unui meniu se găsesc în capitolul dedicat Constructorului de meniuri.

Stabilirea proprietăţilor ferestrei Visual FoxPro

Pentru stabilirea proprietăţilor ferestrei Visual FoxPro se foloseşte variabila sistem _SCREEN, care este de fapt un obiect, cu proprietăţi şi metode. Una dintre proprietăţi este, de exemplu, **Caption**, care stabileşte titlul ferestrei Visual FoxPro. Ca urmare, comanda:

```
_SCREEN.Caption="Noul titlu"
```

schimbă titlul ferestrei în "Noul titlu".

Alte proprietăţi ale acestui obiect pot fi, de asemenea, modificate în programul monitor, pentru a se obţine diferite efecte şi aspecte ale ferestrei Visual FoxPro.

Pornirea procesorului de evenimente

Procesorul de evenimente este activat prin comanda READ EVENTS şi este dezactivat prin comanda CLEAR EVENTS. Prima dintre comenzile amintite trebuie executată după activarea meniului sistemului.

Cea de-a doua comandă trebuie executată atunci când se doreşte terminarea lucrului cu sistemul. Prin urmare, vom include această comandă în secvenţa de cod executată la alegerea opţiunii **Iesire** a meniului sistemului informatic.

Varianta clasică a programului monitor

Să vedem cum asamblăm aceste operaţii într-un program monitor. Varianta clasică a programului monitor este una secvenţială, în care operaţiile precizate anterior sunt executate pe rând. O structură posibilă este cea de mai jos:

```
* Program monitor

* Mai intai se salveaza configurarile curente ale mediului
salv_DELETED = SET("DELETED")
salv_TALK = SET("TALK")
...

* Apoi se stabilesc configurările mediului specifice sistemului
*      informatic
SET DELETED ON
SET TALK OFF
...

* Se stabileste titlul ferestrei principale
_SCREEN.Caption="Aplicatie test"

* Se activează meniul sistemului
DO meniu.mpr

* Se pornește procesorul de evenimente
READ EVENTS

* Se reface vechea stare a mediului
SET DELETED &salv_DELETED
SET TALK &salv_TALK
...
SET SYSMENU TO DEFAULT
```

Desigur că structura de mai sus poate fi îmbunătățită cu diferite elemente specifice sistemului informatic respectiv.

Varianta orientată spre obiecte a programului monitor

Folosindu-se modelul orientării spre obiecte, programul monitor ar putea reprezenta un obiect. Pentru aceasta, trebuie definită o clasă specială, pe baza căreia va fi construit obiectul de tip program monitor. Mai jos este prezentată o variantă a unui astfel de program monitor:

```
prog_mon=CreateObject("c_program","meniu","Aplicatie test")
prog_mon.ProcesorEvenimente

DEFINE CLASS c_program AS custom

  * Aici sunt definite variabilele folosite pentru memorare
  *     care sunt de fapt proprietati ale clasei
  PROTECTED salv_TALK, salv_DELETED, ...
```

```
      * Procedura de initializare
      PROCEDURE Init
        PARAMETERS l_meniu,l_fereastra

        * Se salveaza starea mediului
        This.salv_TALK=SET("TALK")
        This.salv_DELETED=SET("DELETED")
        ...

        * Se stabileste titlul ferestrei (primit din exterior)
        _SCREEN.Caption=l_fereastra
        * Se activeaza meniul
        n_meniu=l_meniu+".MPR"
        DO &n_meniu
        ...

      ENDPROC

      * Procedura de pornire a procesorului de evenimente
      PROCEDURE ProcesorEvenimente
        READ EVENTS
      ENDPROC

      * Procedura de terminare
      PROCEDURE Destroy
        * Se refac parametrii mediului
        sir=This.salv_TALK
        SET TALK &sir
        ...

        * Se revine la meniul sistem al SGBD
        SET SYSMENU TO DEFAULT
      ENDPROC

    ENDDEFINE
```

Observăm că pentru aplicaţie definim o clasă specială (moştenită din **custom** – clasă definită de utilizator). Pentru această clasă am definit o serie de proprietăţi folosite la memorarea parametrilor mediului. Au fost definite metodele **Init** (de iniţializare) şi **Destroy** (de terminare), care sunt apelate automat la crearea, respectiv distrugerea obiectului. De asemenea, a fost definită metoda **ProcesorEvenimente**, care este apelată pentru pornirea procesorului de evenimente al sistemului.

Meniul sistemului conţine o opţiune **Ieşire**, la apelarea căreia este executată comanda `CLEAR EVENTS`.

Proiecte. Gestionarea componentelor unui sistem informatic

Def

***Proiectele** reprezintă o facilitate pusă la dispoziția proiectanților sistemelor informatice, cu ajutorul căreia se ține evidența și sunt coordonate elementele unui sistem informatic, cum ar fi baze de date, tabele, programe, forme, meniuri, rapoarte, interogări etc.*

Proiectele sunt folosite atunci când se lucrează cu un număr mai mare de elemente (programe, tabele,...), pentru a ține evidența acestora. În general, lucrul la un sistem informatic debutează prin crearea fişierului proiect; apoi, ori de câte ori se creează un nou element, acesta este adăugat la proiect. Toate operațiile care se efectuează asupra elementului respectiv se pot declanşa (aşa se şi recomandă) din interiorul proiectului (din fereastra Gestionarului de proiecte), în acest fel proiectul fiind totdeauna înştiințat despre modificărilor efectuate asupra unui element.

Dacă pentru un sistem informatic (în lucru) s-a creat un proiect, atunci sunt posibile o serie de operații, precum:

- actualizarea tuturor versiunilor compilate ale programelor sistemului informatic. Aceasta înseamnă că pentru fiecare program al sistemului se vor compara cele două variante, cea sursă şi cea compilată, şi, dacă există desincronizări, se va declanşa automat compilarea;
- construirea aplicațiilor şi a programelor executabile. Proiectele reprezintă punctul de plecare în construirea aplicațiilor şi a programelor executabile, variante folosite atunci când sistemul informatic este distribuit utilizatorilor.

La nivelul inferior, un proiect reprezintă un fişier cu o structură specială pe discurile sistemului de calcul. Fişierul este de fapt o tabelă (cu extensia implicită `.PJX`) ale cărei înregistrări corespund elementelor componente ale proiectelor. Includerea unui element într-un proiect nu înseamnă încorporarea sa fizică în acesta, ci doar memorarea unor date specifice elementului respectiv (poziția sa pe disc, data şi ora construirii sau a ultimei modificări etc.).

Tipurile de elemente care pot fi incluse într-un proiect sunt date în următorul tabel:

Tipul elementului	Denumirea în engleză
baze de date	**Databases**
tabele izolate	**Free Tables**

interogări	**Queries**
forme	**Forms**
rapoarte	**Reports**
etichete	**Labels**
biblioteci de clase	**Class Libraries**
programe	**Programs**
biblioteci API	**API Libraries**
aplicaţii	**Applications**
meniuri	**Menus**
fişiere de text	**Text Files**
alte tipuri de fişiere	**Other Files**

Pentru gestiunea proiectelor este folosit un utilitar al mediului Visual FoxPro, numit Gestionarul de proiecte. Acest utilitar este pornit prin comanda:

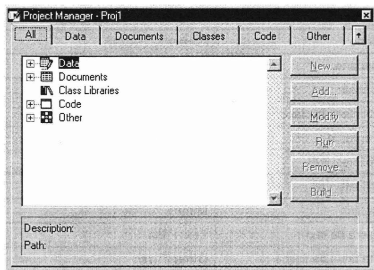
```
MODIFY PROJECT <nume proiect>
```

Dacă proiectul este deja creat, el va fi deschis pentru modificare în fereastra Gestionarului de proiecte, iar dacă nu există pe disc, va fi creat.

Interactiv, crearea unui nou proiect este declanşată prin alegerea opţiunii **New** a meniului **File**. Din fereastra de dialog deschisă pe ecran se alege butonul radio **Project** (proiect) şi apoi se acţionează butonul **New file**. Urmează specificarea numelui noului proiect într-o altă fereastră de dialog, după care proiectul este deschis în fereastra Gestionarului de proiecte.

Dacă proiectul este deja creat, deschiderea sa în fereastra Gestionarului de proiecte se face prin alegerea opţiunii **Open** a meniului **File**, urmată de alegerea opţiunii *Project* (proiect) din lista derulantă **Files of type** (fişiere de tipul...) – indicându-se astfel faptul că se urmăreşte deschiderea unui proiect – şi de acţionarea butonului **OK**.

Indiferent de modul de pornire a Gestionarului de proiecte, fereastra acestuia arată ca în figura următoare:



Fereastra conţine în centru o listă în care sunt afişate elementele componente ale proiectului, grupate pe categorii. În partea stângă a ferestrei se află un grup de butoane, folosite pentru realizarea diferitelor operaţii cu elementele proiectului. Paginile alternative ale ferestrei permit selectarea elementelor de anumite tipuri ale proiectului.

Adăugarea de noi elemente la un proiect

Adăugarea unui nou element la proiect se realizează astfel:

- pentru ca în lista Gestionarului de proiecte să fie afişat tipul elementului, se activează acea pagină alternativă a ferestrei care corespunde tipului respectiv. Din listă se selectează apoi tipul elementului de adăugat.
- dacă elementul este deja creat şi trebuie doar adăugat la proiect, se acţionează butonul **Add** (adăugare), iar din fereastra de dialog deschisă pe ecran se alege elementul respectiv;
- dacă elementul dorit nu este deja creat şi se doreşte crearea şi adăugarea sa la proiect, se acţionează butonul **New** (nou). Va fi pornit astfel Constructorul asociat tipului de element specificat.

O metodă bună de lucru este cea a creării tuturor elementelor noi ale sistemului informatic din interiorul Gestionarului de proiecte, pentru a se evita astfel eventualele omiteri ale unor fişiere.

Editarea unui element al unui proiect

Din Gestionarul de proiecte se poate porni direct Constructorul asociat unui anumit tip de element, pentru editarea (modificarea) unui element al proiectului. Mai întâi se selectează din lista Gestionarului de proiecte elementul vizat şi apoi se acţionează butonul **Modify** (modificare).

Rularea unui element al unui proiect

Rularea unui element (program, formă etc.) al proiectului se realizează prin acţionarea butonului **Run** (rulare), precedată de selectarea elementului de rulat din lista Gestionarului de proiecte.

Înlăturarea unui element din proiect

Pentru a elimina un element din proiectul în curs de editare, elementul respectiv se selectează din lista Gestionarului de proiecte, după care se acţionează butonul **Remove** (înlăturare). Înlăturarea unui element dintr-un proiect nu înseamnă şi ştergerea acestuia de pe disc, ci doar ştergerea indicatorului corespunzător din tabela proiectului.

După acţionarea butonului **Remove**, utilizatorul este consultat, printr-o fereastră de dialog, dacă doreşte ştergerea efectivă de pe disc a fişierului respectiv (butonul **Delete**) sau doar înlăturarea din proiect (butonul **Remove**).

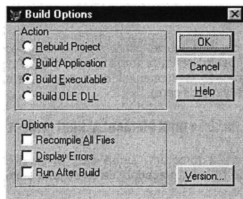
Recompilarea elementelor proiectului

Una dintre facilităţile oferite de un proiect este recompilarea componentelor sale (programe), a tuturor componentelor sau numai a celor modificate de la ultima compilare.

Pentru recompilare, în fereastra Gestionarului de proiecte se află un buton numit **Build** (construire). O dată acţionat acest buton, pe ecran este deschisă fereastra de dialog din figura următoare, care permite specificarea unor opţiuni de recompilare. Butoanele radio din partea de sus a ferestrei sunt folosite pentru specificarea destinaţiei operaţiei, adică a ceea ce se va construi (se vor recompila elementele proiectului, se va crea o aplicaţie, un program executabil sau un fişier executabil de tip DLL).

Secţiunea **Options** (opţiuni) a ferestrei permite specificarea următoarelor opţiuni:

- recompilarea tuturor componentelor proiectului, când se activează comutatorul **Recompile All Files** (recompilarea tuturor fişierelor), sau numai a celor modificate şi necompilate, când comutatorul este dezactivat;
- afişarea erorilor rezultate în urma operaţiei de recompilare. Comutatorul care activează această operaţie este **Display Errors** (afişare erori);



- rularea programului generat imediat după recompilare, dacă este selectat comutatorul **Run After Build** (rulare după construire).

Dacă se dorește doar recompilarea componentelor proiectului, se alege butonul radio **Rebuild Project** (reconstruire proiect) și apoi se acționează butonul **OK**.

Generarea aplicațiilor și a programelor executabile

Generarea aplicațiilor

Pe lângă evidența componentelor unui proiect, Gestionarul de proiecte este folosit și la generarea aplicațiilor și a programelor executabile.

Def

Aplicațiile *reprezintă un tip special de fișiere, care pot fi executate direct de sistem și care conțin programele și celelalte tipuri de elemente ale unui sistem informatic.*

Aplicațiile conțin mai multe programe, forme, rapoarte etc., toate grupate într-un singur fișier de tip aplicație, care are extensia implicită .APP.

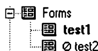
Pentru construirea unei aplicații, se pleacă de la datele memorate în fișierul proiect corespunzător. Deci, înainte de a trece la construirea unei aplicații, este necesară includerea în proiectul respectiv a tuturor elementelor sistemului informatic. În cazul lipsei unui element, este generată o eroare.

Din punctul de vedere al încorporării efective în fișierul aplicație, elementele unui proiect se împart în două grupe:

- elemente care fac parte din proiect şi sunt incluse în fişierul aplicaţie;
- elemente care fac parte din proiect, dar nu sunt incluse în fişierul aplicaţie.

În prima categorie se încadrează programele, formele, rapoartele şi toate acele elemente care nu se modifică la rulare. Cea de-a doua categorie cuprinde elemente care se modifică la rulare, cum ar fi de exemplu tabelele de date, care, evident, se modifică în timpul lucrului în sistemul informatic (se adaugă date, se modifică sau se şterg date). Există însă şi tabele cu parametri care nu se modifică la rulare; acestea pot fi încorporate efectiv în fişierul aplicaţie.

Un element al unui proiect care nu va fi încorporat în fişierul aplicaţie are în dreptul său, în partea stângă, un mic cerculeţ tăiat (ca forma `test2` în figura de mai jos).



Pentru a schimba starea unei componente a unui proiect din încorporabilă în neîncorporabilă se selectează elementul respectiv şi apoi se alege opţiunea **Exclude** (excludere) a meniului **Project**. Pentru transformarea inversă se foloseşte opţiunea **Include** (includere) a aceluiaşi meniu.

Un alt aspect ce trebuie precizat înainte de construirea efectivă a unei aplicaţii este programul principal al sistemului informatic, adică acel program care va fi lansat atunci când se execută comanda de rulare a aplicaţiei. Stabilirea programului principal se face prin selectarea acestuia din lista Gestionarului de proiecte şi alegerea opţiunii **Set Main** (stabilire program principal) a meniului **Project**. În listă, programul principal va fi afişat cu caractere aldine (a se vedea în figura de mai sus forma `test1`).

Pentru generarea unei aplicaţii pe baza proiectului curent se foloseşte butonul **Build** al ferestrei Gestionarului de proiecte. În fereastra deschisă pe ecran se alege butonul radio **Build Application** (construire aplicaţie) şi apoi se acţionează butonul **OK**. În noua fereastră deschisă pe ecran se introduce numele fişierului aplicaţie ce va fi generat.

Rularea unei aplicaţii se face cu comanda DO, fiind însă necesară precizarea explicită a extensiei fişierului:

```
DO <aplicaţie>.APP
```

Generarea programelor executabile

Visual FoxPro este un mediu care permite rularea programelor FoxPro în variantă compilată (`.FXP`) şi a aplicaţiilor (`.APP`). Însă aceste programe pot fi rulate numai în mediul Visual FoxPro şi deci, pentru rularea lor în alt sistem de calcul, este necesară

instalarea în sistemul de calcul respectiv şi a SGBD. Prin facilitatea de construire a programelor executabile se obţine o independenţă a programelor faţă de mediul Visual FoxPro şi distribuirea lor independentă.

Ca şi în cazul aplicaţiilor, construirea unui program executabil este precedată de construirea proiectului corespunzător, în care trebuie incluse toate elementele sistemului informatic. Din nou, trebuie precizate elementele care vor fi încorporate în fişierul executabil şi cele care vor fi lăsate în afară şi, de asemenea, trebuie indicat programul principal, care va fi executat primul la comanda de rulare a programului executabil.

Construirea unui fişier executabil pe baza unui proiect se face la fel ca în cazul aplicaţiei, cu excepţia butonului radio selectat din fereastra opţiunilor de construire, care trebuie să fie **Build Executable** (construire program executabil).

Va fi generat un fişier având acelaşi nume cu al proiectului, dar extensia .EXE. Acest fişier poate fi executat independent de mediul Visual FoxPro, pentru aceasta fiind totuşi necesare o serie de biblioteci de proceduri ale sistemului.

Distribuirea sistemelor informatice

În general, construirea unui sistem informatic are ca scop distribuirea sa către beneficiari, care plătesc pentru acest serviciu. Distribuirea presupune furnizarea către utilizatorul final a tuturor componentelor sistemului informatic. În funcţie de formatul ales pentru distribuire (aplicaţie sau program executabil), programele trebuie însoţite de anumite elemente care să permită rularea lor în sistemul de calcul destinaţie.

În cazul aplicaţiilor, acestea necesită pentru rulare instalarea mediului Visual FoxPro. Există însă o variantă a mediului, numită „pentru rulare", care conţine doar acele elemente necesare pentru rularea programelor, nu şi pe cele pentru proiectarea, compilarea sau depanarea acestora. Prin urmare, în cazul unei aplicaţii, este suficientă furnizarea către utilizatorul final a mediului de rulare Visual FoxPro.

Dacă sistemul este distribuit în formă executabilă, nu mai este necesar mediul Visual FoxPro (nici măcar mediul de rulare). Sunt suficiente bibliotecile de rulare, adică o serie de fişiere care conţin, într-un format specific sistemului Windows, procedurile necesare rulării.

Toate aceste fişiere trebuie depuse de către proiectant pe un suport de transport (dischete, CD etc.), într-un format cât mai compact, şi apoi trecute în sistemul de calcul destinaţie. Alcătuirea setului de fişiere care se copiază pe suportul de transport (a kitului se instalare) se poate face cu un utilitar al sistemului Visual FoxPro numit *Vrăjitorul kitului de instalare* (**Setup Wizard**).

O dată construit kitul de instalare al unui sistem informatic şi trecut pe suportul extern de memorare, se poate instala sistemul informatic în orice sistem de calcul prin

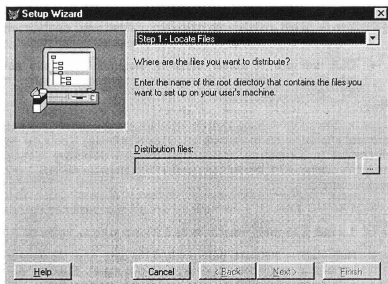
simpla rulare a programului de instalare, care preia toate sarcinile (decompactare, copiere, configurare etc.).

Vrăjitorul kitului de instalare

Construirea kitului de instalare începe cu crearea unei imagini a sistemului informatic în sistemul de calcul în care se lucrează. Această imagine constă, de fapt, în structura de directoare necesară rulării sistemului informatic, structură în care sunt copiate fişierele acestuia (fişierul aplicaţie, fişierul executabil, fişierele externe etc.).

Pe baza acestei imagini sursă, Vrăjitorul va crea o altă imagine, destinaţie, care va corespunde de această dată kitului de instalare. Chiar dacă imaginea destinaţie este creată pe hard-disc, ea va avea structura suportului de memorare specificat în ferestrele Vrăjitorului. De exemplu, dacă se precizează dischetele de 1.44 M, în directorul destinaţie de pe hard-disc se vor crea mai multe directoare de câte 1.44 M, al căror conţinut va fi copiat pe câte o dischetă.

Lucrul cu Vrăjitorul de kituri de instalare implică parcurgerea mai multor ferestre de dialog, precum cea de mai jos:



În fiecare dintre aceste ferestre se precizează o serie de parametri. Trecerea de la un pas la altul se face prin acţionarea butoanelor **Next** (următorul) şi **Back** (înapoi).

Parametrii precizaţi la fiecare pas sunt explicaţi în cele ce urmează:

Pasul 1

- Se precizează locul în care se află imaginea sursă a sistemului informatic (câmpul de editare **Distribution files** – fişierele de distribuit).

Pasul 2

- În acest pas se permite identificarea componentelor speciale folosite de aplicaţie, adică:
  - **Visual FoxPro runtime** – versiunea de rulare a programului Visual FoxPro, necesară atunci când sistemul este distribuit sub formă de aplicaţie;
  - **Microsoft Graph 5.0 runtime** – versiunea de rulare a programului Microsoft Graph 5.0, folosită pentru crearea graficelor. Dacă în sistemul informatic au fost folosite grafice create cu acest program, este necesară activarea comutatorului;
  - **ODBC drivers** – driverele ODBC care pot fi folosite ca surse de date externe. Prin intermediul conexiunilor din Visual FoxPro, se pot folosi date din baze de date de alte tipuri (alte SGBD). Dacă în sistemul informatic este folosită această facilitate, atunci driverele respective trebuie şi ele instalate pe calculatorul destinaţie, operaţie efectuată de viitorul kit de instalare, dacă este activat acest comutator;
  - **OLE servers** – atunci când în sistemul informatic se foloseşte tehnologia serverelor OLE, trebuie activat acest comutator.

Pasul 3

- Locul în care se va depune kitul de instalare creat şi formatul acestuia (dat de formatul suportului de memorare folosit la distribuire). Locul de destinaţie se precizează în câmpul de editare **Disk images directory** (directorul pentru imaginile dischetelor), iar tipul suportului de memorare se indică prin activarea comutatoarelor:
  - **1.44 MB 3.5-inch** – dischete de 3.5 inci şi capacitatea de 1.44 MB;
  - **1.2 MB 5.25-inch** – dischete de 5.25 inci şi capacitatea de 1.2 MB;
  - **Netsetup** – instalare pentru reţea, în care kitul de instalare nu este împărţit în segmente, ci este „dintr-o bucată". Suportul de memorare este de obicei un CD, pe care se depune imaginea kitului.

Pasul 4

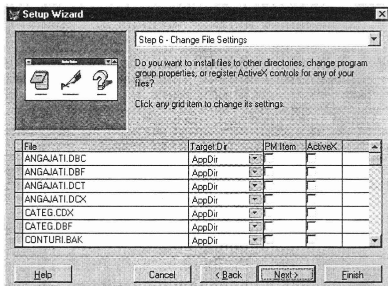
- Titlul ferestrei de dialog folosite de kitul de instalare (**Setup dialog box caption**) şi datele referitoare la drepturile de copyright (**Copyright information**);
- De asemenea, în acest pas se poate specifica un program executabil care să fie lansat în execuţie la sfârşitul instalării. Acest program poate fi folosit pentru protecţia kitului împotriva copierilor ilegale sau pentru realizarea unor configurări specifice sistemului informatic. Câmpul de editare în care se precizează numele şi locul programului de executat este **Post-setup executable**.

Pasul 5

- Directorul implicit în care se va instala sistemul, dacă cel ce rulează kitul nu specifică altceva. Câmpul de editare se numeşte **Default Directory** (director implicit);
- Numele grupului (**Program group**) care va fi adăugat la meniul de start al sistemului de operare (Windows 95).

Pasul 6

- La acest pas se oferă acces la tabela creată pentru memorarea datelor referitoare la fişierele incluse în kitul de instalare. Pentru aceasta se foloseşte o grilă, în care fiecare linie conţine date referitoare la un fişier:



- Coloanele grilei au următoarele semnificaţii:
  - prima coloană este una de butoane cu ajutorul cărora se poate şterge o anumită linie, fişierul respectiv fiind astfel eliminat din kitul de instalare (nu va inclus în kit şi deci nu va ajunge în sistemul de calcul destinaţie);
  - coloana **File** (fişier) precizează numele fişierului respectiv;
  - coloana **Target directory** (director destinaţie) precizează directorul în care va fi copiat fişierul respectiv (*AppDir* – directorul aplicaţiei, *WinDir* – directorul Windows al sistemului de operare, *WinSysDir* – directorul `Windows\System` al sistemului de operare);
  - coloana **PM Item** – dacă se activează acest comutator, va fi pornit programul **Program Manager** al sistemului pentru specificarea unor proprietăţi suplimentare ale fişierului respectiv, cum ar fi, de exemplu, pictograma asociată;
  - coloana **ActiveX** – ca urmare a activării comutatorului din această coloană, kitul de instalare va înregistra controlul de tip ActiveX în sistemul de calcul destinaţie.

Pasul 7

- Acţionarea butonului **Finish** (terminare) va determina începerea procesului de construire a kitului, obţinându-se, în final, imaginea acestuia în directorul destinaţie.

După generarea imaginii kitului de instalare, Vrăjitorul prezintă pe ecran o serie de informaţii generale (numărul de dischete, spaţiul ocupat pe fiecare dischetă etc.).

Tot ce rămâne de făcut pentru a avea efectiv un kit de instalare este copierea imaginii respective pe dischete, care urmează a fi predate utilizatorului. Imaginea se copiază dintr-un subdirector al directorului destinaţie specificat la pasul 3. De exemplu, în cazul unui kit construit pentru dischete de 1.44 MB, în directorul destinaţie va exista directorul `DISK144`, care va avea ca subdirectoare `DISK1`, `DISK2`,..., în fiecare dintre acestea găsindu-se fişierele ce vor fi copiate pe discheta corespunzătoare.

Instalarea efectivă a sistemului informatic pe calculatorul destinaţie se face prin rularea programului `SETUP.EXE` de pe prima dischetă a kitului.

Partea a V-a – TEHNICI SPECIALE

Tehnici speciale disponibile în Visual FoxPro

Capitolul 13

- Tehnologia OLE
  - ✓ Ce este OLE şi la ce foloseşte?
  - ✓ Câmpuri „generale"
  - ✓ Obiectele OLE în forme
- Schimbul dinamic de date între aplicaţii prin DDE
  - ✓ Ce este DDE şi la ce foloseşte?
  - ✓ Visual FoxPro pe post de client DDE
  - ✓ Visual FoxPro ca server DDE
- Aplicaţii client/server
  - ✓ Ce înseamnă client/server?
  - ✓ Construirea vederilor la distanţă
  - ✓ Câteva aspecte legate de proiectarea aplicaţiilor client/server

Tehnologia OLE

Ce este OLE şi la ce foloseşte?

Def

*OLE (**Object Linking and Embedding** – legare şi încorporare de obiecte) reprezintă un mecanism prin care se realizează partajarea datelor între mai multe aplicaţii.*

Cu ajutorul acestei tehnologii, putem introduce într-o tabelă un document, o imagine sau o foaie de calcul tabelar, create în afara SGBD-ului cu alte aplicaţii, precum Word, Excel, Paintbrush etc. Versiuni mai noi ale tehnologiei (protocolului) OLE, cum ar fi 2.0, permit şi comunicarea dinamică între diferite aplicaţii.

Este posibilă afişarea şi editarea documentelor în formele construite în Visual FoxPro, precum şi afişarea obiectelor OLE în rapoartele create cu ajutorul Constructorului de rapoarte.

Câmpuri „generale"

O dată cu evoluţia bazelor de date şi a sistemelor SGBD, s-a diversificat şi gama datelor ce pot fi memorate şi gestionate de SGBD. Metoda prin care s-a realizat acest lucru în Visual FoxPro este cea a câmpurilor „generale".

Câmpurile „generale" reprezintă nişte tipuri speciale de câmpuri, în care pot fi încărcate, cu ajutorul tehnologiei OLE, diverse date, documente sau fişiere create cu alte aplicaţii decât Visual FoxPro.

Exemplu

De exemplu, am putea avea un câmp al unei tabele care să memoreze imagini create cu un program grafic sau documente create cu un program de editare a textelor. O bază de date pentru memorarea informaţiilor referitoare la personalul unei unităţi economice poate să conţină un câmp pentru poza angajatului şi altul pentru caracterizarea fiecărui angajat (în format Word, de exemplu).

Conţinutul câmpurilor „generale" este construit cu o altă aplicaţie decât Visual FoxPro (editor de texte, program de calcul tabelar, program de desenare etc.), şi este ataşat câmpului prin tehnologia OLE.

Există două moduri prin care se stabileşte legătura între câmpul general al tabelei şi conţinutul acestuia:

- **prin încorporare** – caz în care conţinutul respectiv este inclus efectiv în tabelă, într-un mod analog cu cel al câmpurilor „memo";
- **prin legare** – când conţinutul nu este inclus în tabelă, ci este doar ataşat acesteia. De fapt, în tabelă se memorează o trimitere spre fişierul în care se află datele respective, fişier creat anterior cu un program corespunzător tipului de date.

Diferenţa dintre cele două metode este dată de locul datelor respective (în tabelă sau în afara ei). Din această cauză, transferul (copierea sau mutarea) unei tabele cu un câmp general la care s-au legat diferite documente trebuie efectuat împreună cu documentele respective.

Fiecare tip de document care poate fi încorporat sau legat la un câmp „general" are asociat programul cu care a fost creat şi care este folosit la modificarea acestuia. În Windows există o bază de date care stabileşte programul cu care este prelucrat fiecare tip de fişier; aceasta este utilizată atunci când conţinutul câmpurilor „generale" se modifică.

Prelucrarea interactivă a câmpurilor „generale"

Pentru încărcarea într-un câmp de tip general a unui obiect prin tehnica OLE, vom proceda astfel:

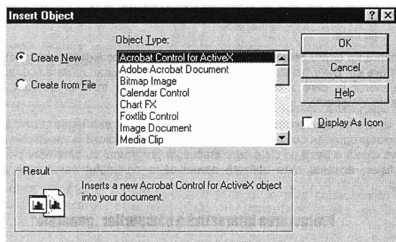
- mai întâi vom deschide tabela care conţine câmpul şi apoi, pentru tabela respectivă, vom deschide o fereastră de editare de tip **Browse**:

Personal

Prenume	Functie	Poza
Gabriel	Dir	gen
Dumitru	Age	gen
Vasile	Dir	gen
Constantin	Dir	gen

- urmează deplasarea cursorului în câmpul de tip „general" şi acţionarea combinaţiei de taste Ctrl+PgDn sau execuţia cu mouse-ul a unui clic dublu pe acel câmp. Ca urmare a acestor operaţii, va fi deschisă o fereastră în care va fi afişat conţinutul câmpului „general" respectiv;
- adăugarea în câmp a unui obiect se face prin alegerea opţiunii **Insert Object** (inserare obiect) a meniului **Edit** (editare). Pe ecran este deschisă o fereastră

de dialog pentru specificarea tipului de obiect şi, implicit, a aplicaţiei cu care este creat acesta;



- mai întâi se specifică dacă obiectul urmează să fie creat (butonul **Create New** – creare obiect nou) sau să fie preluat dintr-un fişier creat anterior (**Create from File** – creat din fişier);
- dacă obiectul va fi creat pe loc, este necesară specificarea tipului de obiect (şi deci a aplicaţiei sursă) prin alegerea din lista **Object Type** (tip obiect);
- dacă obiectul va fi preluat dintr-un fişier creat anterior, atunci este necesară specificarea numelui şi locului acestuia în câmpul de editare **File** (fişier). Obiectul din fişier va fi legat la tabelă, dacă se activează comutatorul **Link** (legare), sau va fi încorporat în tabelă, când comutatorul este dezactivat;
- acţionarea butonului **OK** fie încorporează obiectul din fişier în tabelă, fie porneşte aplicaţia sursă cu care se creează obiectul respectiv (ieşirea din aplicaţie determină încorporarea obiectului nou creat);
- fereastra câmpului „general" se poate închide cu combinaţia de taste Ctrl+F4, după care se revine în fereastra **Browse**.

Modificarea unui obiect introdus anterior într-un câmp „general" se face astfel:

- din fereastra **Browse**, se deschide fereastra câmpului „general". Printr-un clic dublu sau prin apăsarea tastei Enter se intră în editarea obiectului respectiv, folosindu-se aplicaţia sursă corespunzătoare;

- ieşirea din aplicaţia sursă determină revenirea în fereastra câmpului „general". Închiderea cu salvare a acestei ferestre se face prin Ctrl+F4 (combinaţia standard din Windows pentru închiderea ferestrelor), iar fără salvare, cu tasta Escape.

Ştergerea unui obiect OLE dintr-un câmp se face prin alegerea opţiunii **Clear** (ştergere) din meniul **Edit**, atunci când este deschisă fereastra câmpului „general" respectiv.

Prelucrarea prin cod a câmpurilor „generale"

Operaţiile prezentate mai sus se pot executa şi cu ajutorul comenzilor limbajului FoxPro. Comanda pentru încărcarea unui câmp „general" cu un anumit document este `APPEND GENERAL`:

```
APPEND GENERAL <câmp general> FROM <fişier> LINK
```

Această comandă face ca în câmpul „general" specificat al înregistrării curente să fie încorporat (dacă lipseşte clauza `LINK`) sau să fie legat (în prezenţa clauzei `LINK`) obiectul din fişierul precizat în clauza `FROM`. Dacă în câmp este deja încărcat un obiect, el va fi înlocuit cu cel din fişierul specificat.

Dacă se doreşte modificarea unui obiect OLE dintr-un câmp „general", se foloseşte comanda `MODIFY GENERAL`:

```
MODIFY GENERAL <câmp general>
```

Ca urmare a acestei comenzi, pe ecran va fi deschisă o fereastră în care va fi afişat obiectul OLE şi din care se va putea porni aplicaţia sursă pentru modificarea acestuia.

Exemplu

*Următoarea secvenţă de cod încarcă o imagine bitmap din fişierul `IMAG.BMP` în câmpul general `POZA` al tabelei `PERSONAL` şi apoi deschide o fereastră în care afişează imaginea respectivă. Din această fereastră se poate porni aplicaţia **Paintbrush** pentru modificarea imaginii (printr-un clic dublu pe imagine).*

```
USE personal
GOTO TOP
APPEND GENERAL poza FROM imag.bmp LINK
MODIFY GENERAL poza
CLOSE ALL
```

Obiectele OLE în forme

Tehnologia OLE se poate folosi şi pentru introducerea într-o formă a unor obiecte create cu alte aplicaţii: documente, foi de calcul tabelar, imagini etc. Pentru aceasta, Constructorul de forme este prevăzut cu două tipuri speciale de obiecte de interfaţă:

- obiectele OLE asociate câmpurilor „generale" şi
- obiectele OLE container.

Primele tipuri de obiecte, adică cele asociate câmpurilor „generale", permit afişarea şi editarea într-o formă a conţinutului unui câmp de tip „general". În schimb, obiectele OLE container permit afişarea şi editarea diferitelor obiecte de tip OLE, independent de vreun câmp „general" al unei tabele. După cum sugerează şi numele, ele reprezintă un container pentru obiectul respectiv – obiectul OLE este memorat direct în formă.

Când se doreşte editarea unui câmp general într-o formă, se foloseşte primul tip de obiect OLE de interfaţă. Când se doreşte însă afişarea (şi, eventual, modificarea) într-o formă a unui obiect OLE, acelaşi la fiecare deschidere a formei, fără memorarea sa într-o tabelă, se foloseşte obiectul OLE container.

Exemplu

De exemplu, în cadrul unui sistem informatic putem construi o formă pentru afişarea informaţiilor de ajutor (help). Pentru a obţine un aspect cât mai plăcut, putem să stocăm aceste informaţii într-un fişier Word (tehnoredactat corespunzător) şi să introducem conţinutul acestuia în formă ca un obiect OLE container. Nu s-a folosit obiectul OLE asociat unui câmp general, deoarece conţinutul sistemului help este acelaşi la fiecare deschidere a formei de ajutor (este vorba de un ajutor independent de context).

Ca şi celelalte obiecte de interfaţă ale formelor, obiectele OLE au proprietăţi şi metode specifice. Câteva dintre cele mai importante sunt următoarele:

- **ControlSource** – valabilă numai în cazul obiectelor OLE asociate câmpurilor „generale", permite specificarea câmpului (de tip „general") folosit ca sursă de date pentru obiect;
- **AutoActivate** – specifică evenimentul care declanşează intrarea în editarea obiectului (lansarea în execuţie a aplicaţiei sursă a obiectului OLE). Acest eveniment ar putea fi:
    - *0 – Manual* – controlul nu este activat automat, ci operaţia trebuie să fie realizată prin cod, folosindu-se metoda **DoVerb** (a se vedea mai jos);
    - *1 – GotFocus* – activarea obiectului are loc atunci când acesta devine ţinta intrărilor;

- *2 – Double click* – evenimentul care declanşează activarea obiectului este clicul dublu pe obiect sau apăsarea tastei Enter când obiectul este ţinta intrărilor;
- *3 – Automatic* – evenimentul declanşator al activării obiectului este cel implicit al tipului de obiect OLE;

- **AutoSize** – în cazul *adevărat*, face ca zona din formă alocată obiectului să fie ajustată corespunzător dimensiunii obiectului sursă;
- **Sizable** – dacă are valoarea *adevărat*, permite utilizatorului să redimensioneze dinamic (la rulare) zona din formă alocată obiectului;
- **Stretch** – precizează modul în care sunt ajustate dimensiunile obiectului OLE în funcţie de dimensiunile zonei din formă rezervate obiectului de interfaţă (a se vedea obiectul de interfaţă de tip imagine, în capitolul referitor la Constructorul de forme).

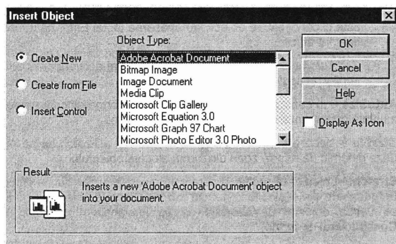
O metodă importantă a unui obiect OLE este **DoVerb**, cu ajutorul căreia sunt executate prin cod diferite operaţii cu obiectul respectiv. Codul operaţiei de executat este furnizat ca parametru metodei **DoVerb**. În general, operaţiile ce pot fi executate cu un anumit obiect sunt dependente de tipul obiectului respectiv. Cu toate acestea, există o serie de operaţii standard, cum ar fi executarea acţiunii implicite a obiectului (pentru care parametrul transmis este 0), activarea obiectului pentru editare (parametrul –1), deschiderea unei ferestre speciale pentru editarea obiectului (valoarea –2) etc.

Introducerea într-o formă a unui obiect OLE asociat unui câmp general se face prin acţionarea butonului de pe bara utilitară a obiectelor de interfaţă a Constructorului de forme, urmată de trasarea pe formă a zonei ce va fi ocupată de obiectul respectiv. După realizarea acestei operaţii, se poate trece la precizarea proprietăţilor şi a metodelor specifice obiectului (precum **ControlSource**).

Butonul corespunzător pentru obiectele OLE container este . După acţionarea sa şi trasarea zonei din formă rezervate obiectului, este deschisă fereastra de dialog pentru specificarea tipului de obiect OLE (a se vedea figura următoare).

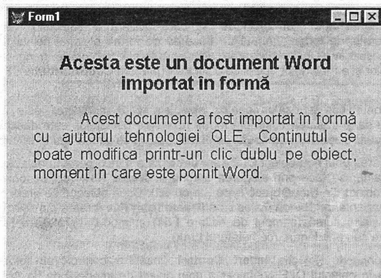
Crearea unui nou obiect OLE care urmează a fi încorporat în obiectul de interfaţă OLE container se face prin alegerea butonului **Create New** (creare obiect nou) şi a tipului de obiect din lista **Object Type** (tip obiect). Dacă obiectul se preia dintr-un fişier (a fost creat anterior), butonul ales va fi **Create from File** (creare din fişier). Mai trebuie precizat fişierul sursă (câmpul de editare **File**) şi modul de ataşare la formă, prin încorporare sau prin legare (comutatorul **Link**).

Butonul de selecţie **Insert Control** (inserare control) se foloseşte pentru includerea în obiectul OLE container a unui obiect de interfaţă de tip ActiveX (pentru aceste obiecte se foloseşte tot tehnologia OLE).



Exemplu

În forma de mai jos a fost definit un obiect OLE container, în care s-a încărcat un document Word.



Schimbul dinamic de date între aplicații prin DDE

Ce este DDE și la ce folosește?

Def

***DDE (Dynamic Data Exchange** – schimb dinamic de date) reprezintă un protocol cu ajutorul căruia se realizează un schimb de date între două aplicații aflate în execuție.*

DDE nu este implementat numai la nivelul SGBD Visual FoxPro, ci la nivelul sistemului de operare Windows. Și alte aplicații Windows pot comunica între ele prin DDE, de exemplu Word, Excel etc.

Prin DDE se poate realiza, de exemplu, încărcarea datelor într-o foaie de calcul tabelar Excel direct dintr-un program Visual FoxPro, se poate completa și formata un document Word dintr-un asemenea program sau se poate completa o tabelă Visual FoxPro din Excel.

Protocolul DDE funcționează pe bază de mesaje, care sunt transferate între aplicațiile implicate în schimbul de date. O comunicare bazată pe acest protocol este numită **conversație** și are loc între două aplicații, una server și una client. Aplicația care solicită date se numește **client DDE**, iar cea care le furnizează se numește **server DDE**.

O aplicație server DDE poate transmite date la mai mulți clienți DDE, iar un client DDE poate solicita date de la mai multe aplicații server DDE. O aplicație poate fi atât client DDE, cât și server DDE, dar în acest caz sunt necesare două conversații diferite. Sarcina inițierii unei conversații revine clientului DDE.

O conversație este unic determinată prin numele aplicației server DDE și prin **subiectul conversației**. În aplicațiile bazate pe fișiere, subiectul conversației îl constituie numele fișierului (al documentului, al foii de calcul etc.) care conține datele solicitate. În cadrul fișierului, datele solicitate vor fi identificate printr-un articol al cărui nume va fi specific aplicației. De exemplu, în Excel, acesta poate fi identificatorul unei celule (`R1C1`).

Visual FoxPro poate funcționa atât ca server DDE, cât și în postura de client DDE; mai întâi prezentăm postura de client DDE.

Visual FoxPro pe post de client DDE

În postura de client DDE, Visual FoxPro inițiază o comunicație cu un server DDE, de la care solicită diferite date sau spre care trimite diferite comenzi. Modul de alcătuire a unui program în care Visual FoxPro este client DDE este următorul:

- mai întâi se deschide un canal de comunicație între clientul DDE Visual FoxPro și aplicația server DDE, folosindu-se funcția `DDEInitiate()`;
- apoi, clientul DDE poate solicita date de la serverul DDE, prin intermediul funcțiilor `DDERequest()` și `DDEAdvise()`, sau îl poate controla în privința efectuării diferitelor operații, folosind funcția `DDExecute()`;
- după terminarea conversației, canalul trebuie închis, acțiune pentru care se folosește funcția `DDETerminate()`.

Deschiderea canalului se face cu ajutorul funcției `DDEInitiate()`:

```
DDEInitiate(<serviciu>,<subiect>)
```

Aceasta returnează o valoare numerică reprezentând numărul canalului (care va fi folosit ulterior pentru identificarea sa în celelalte funcții de comunicație), dacă deschiderea s-a realizat cu succes, și –1, dacă deschiderea a eșuat. Dacă aplicația server DDE nu se află în execuție, utilizatorul este interogat în legătură cu lansarea ei.

`<serviciu>` reprezintă numele serviciului furnizat de aplicația server DDE, care, de obicei, este numele fișierului executabil al aplicației server. `<subiect>` reprezintă numele subiectului în legătură cu care se realizează transferul de date și este specific aplicației server DDE.

După ce a fost realizat schimbul de date între aplicații, o conversație DDE trebuie încheiată cu ajutorul funcției `DDETerminate()`:

```
DDETerminate(<număr canal>)
```

Dacă operația reușește, `DDETerminate()` returnează *adevărat*, iar în caz contrar returnează valoarea *fals*.

Exemplu

Structura folosită pentru realizarea unei comunicări între Visual FoxPro și o altă aplicație server DDE, în acest caz Excel, este următoarea:

```
* Se deschide un canal de comunicatie intre Visual FoxPro
* si foaia de calcul Tabel1, deschisa in Excel
canal=DDEInitiate('Excel','Tabel1')
```

```
IF canal !=-1
  * aici pot fi folosite functii DDE
  * pentru transferul datelor intre Visual FoxPro si Excel
  ...
  =DDETerminate(canal)    && se inchide canalul
ENDIF
```

Solicitarea datelor de la serverul DDE, după deschiderea unui canal, se realizează cu funcția `DDERequest()`:

```
DDERequest(<canal>,<articol>,<format>,<funcţie>)
```

Funcția returnează un şir de caractere reprezentând datele solicitate, dacă transferul reuşeşte, sau şirul vid în caz de eşec.

`<articol>` reprezintă elementul din care se preiau datele solicitate; tipul său depinde de aplicația server (de exemplu, celulă în cazul programelor de calcul tabelar).

Formatul în care sunt trimise datele solicitate va fi specificat prin `<format>`. Implicit, acesta este `CF_TEXT`, în care câmpurile sunt delimitate prin caractere Tab, iar înregistrările prin `LF` (**Line Feed**, `CHR(10)`) şi `CR` (**Carriage Return**, `CHR(13)`). Mai pot fi folosite formatele `CF_BITMAP`, pentru fişiere de tip bitmap (cu extensia implicita `.BMP`), şi `CF_METAFILEPICT`, pentru metafişiere Windows.

Modul în care se realizează transferul datelor între client şi server depinde de parametrul `<funcţie>`. Dacă `<funcţie>` este omis, transferul datelor va fi sincron. Aceasta înseamnă că, după execuția funcției `DDERequest()`, programul aşteaptă un interval de timp primirea datelor solicitate de la server. Dacă după acest interval clientul nu recepționează datele solicitate (aplicația server este ocupată cu execuția altor operații), programul continuă execuția cu instrucțiunea de după `DDERequest()`. Mărimea intervalului de timp poate fi specificată de utilizator prin funcția `DDESetOption()`, care va fi prezentată mai târziu.

Exemplu

Un exemplu simplu de transfer sincron este prezentat mai jos:

```
canal=DDEInitiate('Excel','Book1')
IF canal !=-1
  solicitare=DDERequest(canal,'R1C1')
  ? solicitare
  =DDETerminate(canal)
ENDIF
```

*În acest exemplu sunt citite datele din celula `R1C1` a tabelului **Book1** deschis anterior în Excel. Observăm lipsa parametrilor trei şi patru (motiv pentru care transferul este sincron).*

Pe ecran va fi afişată valoarea variabilei **`solicitare`**, *adică datele solicitate.*

Un transfer asincron de date poate fi realizat atunci când parametrul `<funcţie>` este prezent în apelul funcţiei. În acest caz, `<funcţie>` trebuie să fie un şir de caractere reprezentând numele unei proceduri sau funcţii definite de utilizator, care va fi executată atunci când vor fi disponibile datele solicitate de la server. Programul nu va aştepta primirea datelor, ci va continua execuţia imediat după lansarea solicitării, chiar dacă nu primeşte datele respective.

În ceea ce priveşte procedura sau funcţia executată la primirea datelor, acesteia îi sunt transmişi şase parametri, cu semnificaţia următoare (în ordine):

- numărul canalului;
- acţiunea – `XACTCOMPLETE` dacă transferul a reuşit şi `XACTFAIL` dacă transferul a eşuat;
- articolul – numele articolului din care au fost preluate datele;
- datele solicitate;
- formatul datelor;
- numărul tranzacţiei.

Exemplu

```
CLEAR
canal=DDEInitiate('Excel','Book1')
IF canal != -1
  solicitare=DDERequest(canal,'R1C1','CF_TEXT','afisare')
  ? solicitare
  WAIT WINDOW "Apasati o tasta pentru continuare"
  =DDETerminate(canal)
ENDIF

PROCEDURE afisare
  PARAMETERS canal, actiune, articol, date, format, nrtranz
  ? 'Canal     '+STR(canal)
  ? 'Actiune  '+actiune
  ? 'Articol  '+articol
  ? 'Date      '+date
  ? 'Format   '+format
  ? 'Nr.tranz.  '+STR(nrtranz)
```

În acest exemplu este solicitat conţinutul celulei `R1C1` *din foaia de calcul* ***Book1*** *(aplicaţia Excel). Dacă datele sunt recepţionate, ele vor fi afişate pe ecran prin intermediul procedurii* **`afisare`**.

```
              0
Canal             0
Actiune XACTCOMPLETE
Articol R1C1
Date    2223

Format  CF_TEXT
Nr.tranz.           0
```

Datele solicitate sunt disponibile atât în variabila `solicitare`*, cât şi în parametrul* `date`*. Dar instrucţiunea* `? solicitare` *determină afişarea valorii 0, deoarece în acel moment datele nu au fost primite de la server.*

O conexiune realizată prin intermediul funcţiei `DDERequest()` este o **conexiune rece**, pentru că transferurile de date se realizează numai la solicitarea clientului. Neajunsul acestui tip de conexiune este imposibilitatea informării clientului DDE cu privire la eventualele modificări ulterioare ale datelor solicitate.

Rezolvarea acestei probleme este dată de **conexiunea caldă**, prin care clientul DDE este informat cu privire la modificările efectuate asupra datelor solicitate. O astfel de conexiune este realizată în Visual FoxPro cu ajutorul funcţiei `DDEAdvise()`:

```
DDEAdvise (<canal>, <element>, <funcţie>, <tip conexiune>)
```

Cu ajutorul acestei funcţii, pot fi stabilite două tipuri de conexiuni calde: **cu notificare** sau **automate**. În primul caz, al conexiunii calde cu notificare, serverul DDE va informa clientul DDE despre modificarea elementului specificat. În cazul unei conexiuni calde automate, pe lângă informarea clientului cu privire la modificarea datelor, se realizează automat şi transmiterea datelor respective (de la server la client).

Specificarea tipului de conexiune caldă se face prin intermediul parametrului `<tip conexiune>`, care va avea valoarea 1 în cazul unei conexiuni cu notificare şi 2 pentru o conexiune automată.

În cadrul aceleiaşi aplicaţii pot fi folosite toate cele trei tipuri de conexiuni: rece, caldă cu notificare şi caldă automată.

Cel de-al treilea parametru transmis funcţiei poate fi folosit pentru specificarea numelui unei funcţii definite de utilizator, care va fi executată la modificarea elementului `<element>`. La apelare, funcţiei îi sunt transmişi şase parametri, în ordine:

- numărul canalului prin care se realizează conexiunea;
- tipul acţiunii – poate lua valorile `ADVISE` sau `TERMINATE`;
- numele articolului;

- date – în cazul unei legături automate, conţine datele solicitate modificate, iar într-o legătură cu notificare conţine şirul vid;
- formatul datelor;
- tipul legăturii – 1 pentru legătură cu notificare şi 2 pentru legătură automată.

Dacă este stabilită o conexiune cu notificare, la modificarea valorii articolului specificat va fi executată funcţia definită de utilizator, iar cel de-al patrulea parametru va conţine şirul vid. În cazul unei conexiuni automate, acest parametru va conţine valoarea modificată a elementului specificat.

Parametrul acţiunii are valoarea ADVISE când conexiunea este actualizată de serverul DDE, respectiv TERMINATE când conexiunea este închisă de server sau de client. Valorile returnate de funcţia definită de utilizator sunt ignorate.

Dacă tipul conexiunii este 0, notificarea va fi inhibată, aşa încât funcţia nu va mai fi executată la modificarea articolului.

Exemplu

*În următorul exemplu se creează o conexiune cu notificare şi una automată. Programul îşi întrerupe execuţia după afişarea unei ferestre de avertizare, pentru a permite utilizatorului să modifice conţinutul celulelor R1C1 şi R1C2. La fiecare modificare a acestora va fi executată procedura **afisare**, care afişează pe ecran conţinutul celor două celule.*

Pentru observarea modului de funcţionare a programului, se recomandă aranjarea pe ecran a ferestrelor Visual FoxPro şi Excel astfel încât ambele să fie vizibile simultan.

```
CLEAR
canal=DDEInitiate('Excel','Book1')
IF canal != -1
   * se stabileste o conexiune cu notificare
   =DDEAdvise(canal,'R1C1','afisare',1)
   WAIT WINDOW 'Modificati celula R1C1 a foii de calcul Book1'
   * se stabileste o conexiune automata
   =DDEAdvise(canal,'R1C2','afisare',2)
   WAIT WINDOW 'Modificati celula R1C2 a foii de calcul Book1'
ENDIF

FUNCTION afisare
   PARAMETERS canal, actiune, articol, date, format, stare
   IF actiune=='ADVISE'
     DO CASE
       CASE articol='R1C1'
         ? 'R1C1 - Conexiune cu notificare '+;
                DDERequest(canal,articol)
```

```
      CASE articol='R1C2'
         ? 'R1C2 - Conexiune automata '+date
   ENDCASE
ELSE
   =DDEAdvise(canal,'R1C1','afisare',0)
   =DDEAdvise(canal,'R1C2','afisare',0)
   =DDETerminate(canal)
ENDIF
```

O funcţie Visual FoxPro care poate fi folosită pentru transmiterea datelor de la clientul DDE la serverul DDE este `DDEPoke()`:

```
DDEPoke(<canal>, <articol>, <date>, <format>, <funcţie>)
```

Ea returnează valoarea *adevărat* dacă datele au fost transmise cu succes şi *fals* în caz contrar.

Pentru a realiza un transfer asincron, prin ultimul parametru se specifică un nume de funcţie definită de utilizator. În acest caz, execuţia programului client DDE continuă imediat ce a fost făcută solicitarea. Funcţia definită de utilizator va fi executată la primirea datelor de către server, acesteia fiindu-i transmişi şase parametri. Cu excepţia ultimului, care conţine numărul tranzacţiei – returnat de `DDEPoke()` – semnificaţiile parametrilor sunt aceleaşi ca în cazul funcţiei `DDERequest()`. Dacă ultimul parametru este omis, aplicaţia client DDE aşteaptă un interval de timp specificat prin `DDESetOption()`.

Exemplu

```
CLEAR
canal=DDEInitiate('Excel','Book1')
IF canal != -1
   =DDEPoke(canal,'R1C1','Randul 1')
   =DDEPoke(canal,'R2C1','Randul 2')
   WAIT WINDOW 'Apasati orice tasta pentru continuare'
   =DDETerminate(canal)
ENDIF
```

Prin intermediul funcţiilor DDE pot fi transmise atât date, cât şi comenzi. Transmiterea unei comenzi către altă aplicaţie poate fi realizată cu ajutorul funcţiei `DDEExecute()`:

```
DDEExecute(<canal>,<comandă>,<format>,<funcţie>)
```

Funcţia returnează *adevărat* dacă aplicaţia primeşte comanda şi o execută cu succes, iar dacă apare o eroare în acest proces, funcţia returnează valoarea *fals*. Semnificaţia parametrilor este asemănătoare cu cea de la funcţiile prezentate anterior.

În cazul unui transfer asincron, funcția specificată este apelată cu cinci parametri; față de cei șase prezentați la funcțiile anterioare, este omis formatul datelor (care nu mai are sens).

Exemplu

Următorul program trimite spre Excel comanda de maximizare a ferestrei sale și apoi selectează prima celulă, R1C1, a foii de calcul Book1.

```
CLEAR
canal=DDEInitiate('Excel','Book1')
IF canal != -1
  =DDEExecute(canal,'[App.Maximize]')
  =DDEExecute(canal,'[Select("R1C1")]')
  =DDETerminate(canal)
ENDIF
```

Caracteristicile schimbului de date între aplicații prin DDE sunt stabilite cu ajutorul funcției `DDESetOption()`:

```
DDESetOption(<parametru>,<valoare>)
```

Parametrul poate fi `TIMEOUT`, caz în care prin valoare se precizează numărul de milisecunde în care funcțiile DDE client așteaptă pentru răspunsul serverului DDE, sau `SAFETY`, când valoarea *adevărat* face ca inițierea eșuată a unui canal de comunicație cu serverul să fie însoțită de o fereastră de dialog de pornire manuală a serverului DDE dorit. Dacă se omite `<valoare>`, funcția returnează valoarea curentă a parametrului specificat.

Întreruperea unei tranzacții asincrone se poate face cu ajutorul funcției `DDEAbortTrans()`:

```
DDEAbortTrans(<număr tranzacție>)
```

Dacă aceasta va fi apelată înainte ca serverul să răspundă, funcția care tratează evenimentul primirii răspunsului de la server nu va fi apelată. `DDEAbortTrans()` returnează o valoare logică: *adevărat* dacă tranzacția asincronă a fost întreruptă și *fals* dacă nu s-a reușit întreruperea.

Pentru a proteja date importante și pentru a întrerupe legături pe perioade scurte de timp, utilizatorul are la dispoziție funcția `DDEEnabled()`. Folosind această funcție, se poate bloca și debloca schimbul dinamic de date printr-un anumit canal sau pentru întregul proces de schimb. Când schimbul de date este blocat, cererile clienților sunt plasate într-o coadă de așteptare până când procesul se deblochează.

Blocarea întregului schimb de date DDE se face prin construcția:

```
=DDEEnabled(.F.)
```

iar deblocarea prin:

```
=DDEEnabled(.T.)
```

Funcţia returnează *adevărat* dacă schimbul de date DDE este deblocat şi *fals* în caz contrar. Dacă funcţia este folosită fără argumente, ea returnează starea curentă a lucrului cu DDE.

Funcţia se poate aplica şi unui anumit canal, caz în care trebuie specificat înainte de valoarea logică respectivă un nou parametru, care să indice numărul canalului vizat.

Erorile apărute la executarea diferitelor operaţii DDE sunt returnate de funcţia `DDELastError()`. Codurile de eroare furnizate de funcţie reprezintă rezultatul ultimei operaţii DDE şi sunt prezentate în următorul tabel:

Cod	Semnificaţie
0	Fără eroare
1	Serviciu ocupat
2	Subiect ocupat
3	Canal ocupat
4	Serviciu inexistent
5	Subiect inexistent
6	Canal inoperant
7	Insuficientă memorie
8	Aşteptare pentru recunoaştere
9	Aşteptare la o cerere de date
10	Nu a fost definit canalul
11	Tranzacţie server aşteptată de client
12	Aşteptare pentru execuţie
13	Parametru invalid
14	Memorie insuficientă
15	Eroare de memorie
16	Conectare eşuată
17	Cerere eşuată

18	Aşteptare pentru transmitere de date
19	Nu se poate afişa mesajul
20	Tranzacţii asincrone multiple
21	Server indisponibil
22	Eroare internă DDE
23	Aşteptare datorată funcţiei `DDEAdvise()`
24	Identificator de tranzacţie invalid
25	Necunoscut

Visual FoxPro ca server DDE

Visual FoxPro poate funcţiona şi ca server DDE, pentru aceasta fiind însă necesare o serie de operaţii preliminare:

- înregistrarea în sistemul de operare Windows a aplicaţiei Visual FoxPro ca server DDE;
- definirea serviciilor furnizate de Visual FoxPro ca server DDE şi a subiectelor conversaţiilor.

Înregistrarea programului Visual FoxPro ca server DDE se realizează cu ajutorul funcţiei `DDESetService()`:

```
DDESetService(<nume server>,'DEFINE')
```

Exemplu

De exemplu, comanda

```
=DDESetService("VFP_DDE","DEFINE")
```

înregistrează serverul cu numele `VFP_DDE` *ca server DDE disponibil.*

După înregistrarea serverului, trebuie definite serviciile furnizate de acesta. Funcţia folosită este tot `DDESetService()`:

```
DDESetService(<server>,<parametru>,<valoare>)
```

`<server>` reprezintă numele folosit la înregistrarea serverului DDE. Parametrul de apel al funcţiei poate fi:

Cod	Semnificaţie
DEFINE	Creează un nou server.
RELEASE	Şterge un server.
ADVISE	Activează sau dezactivează serviciul de informare a clientului DDE în legătură cu modificarea datelor.
EXECUTE	Activează sau dezactivează serviciul de execuţie a comenzilor clientului DDE.
POKE	Activează sau dezactivează serviciul de primire a datelor trimise de către clientul DDE.
REQUEST	Activează sau dezactivează posibilitatea de a primi cereri de date de la clientul DDE.
FORMATS	Specifică formatul datelor.

Instalarea serviciilor disponibile la serverul specificat se face printr-un apel al funcţiei `DDESetService()`, în care `<valoare>` este *adevărat*, `.T.`.

Exemplu

De exemplu, comanda următoare activează serviciul ADVISE al serverului VFP_DDE:

```
=DDESetService('VFP_DDE','ADVISE',.T.)
```

Dacă cel de-al treilea parametru se omite din apelul funcţiei, va fi returnată starea curentă a serviciului specificat.

Exemplu

```
? DDESetService('VFP_DDE','EXECUTE')
```

Ştergerea unui server definit se face cu ajutorul construcţiei:

```
=DDESetService(<server>,'RELEASE')
```

Crearea şi ştergerea subiectelor acceptate de serverul DDE se face prin intermediul funcţiei `DDESetTopic()`:

```
DDESetTopic(<server>,<subiect>,<funcţie>)
```

Această funcție returnează o valoare de tip logic ce indică dacă definirea subiectului s-a realizat cu succes. Dacă `<funcție>` se omite, subiectul respectiv este șters. O dată cu definirea subiectului se stabilește și modul în care serverul DDE va răspunde la solicitarea clientului DDE referitoare la acel subiect. Aceasta se realizează prin includerea în apelul funcției `DDESetTopic()` a celui de-al treilea parametru, care reprezintă funcția definită de utilizator ce va fi apelată pentru a răspunde la solicitarea clientului DDE.

Funcția respectivă este apelată cu șase parametri, în următoarea ordine: număr canal, acțiune, articol, date, format și tipul legăturii.

Exemplu

Prin execuția programului de mai jos se creează serverul `VFP_DDE`. În acest scop se definește subiectul `subiect`, care are asociată procedura `tratare`:

```
=DDESetService('VFP_DDE','DEFINE')
=DDESetService('VFP_DDE','EXECUTE',.T.)
=DDESetTopic('VFP_DDE','subiect','tratare')
WAIT WINDOW "Apasati o tasta pentru terminarea lucrului"
=DDESetService('VFP_DDE','RELEASE')

PROCEDURE tratare
  PARAMETERS canal, actiune, articol, date, format, mesaj
  DO CASE
    CASE actiune='EXECUTE'
      &date
    CASE actiune='TERMINATE'
      ? "Terminare executie"
  ENDIF
```

După ce a fost executat acest program, Visual FoxPro poate fi apelat ca server DDE de un client DDE (de exemplu, din Excel sau Word).

Aplicații client/server

Ce înseamnă client/server?

Răspândirea tot mai mare a rețelelor de calculatoare a impus dezvoltarea concomitentă a diverselor tehnici de comunicație, de partajare a datelor, de utilizare în comun a diferitelor elemente hardware și software. Client/server este una dintre aceste tehnologii care vin să rezolve o serie de probleme apărute în utilizarea rețelelor de calculatoare. Modelul client/server stabilește un nou mod de construire a aplicațiilor în

rețea, având ca efect scăderea semnificativă a cantității de date vehiculate pe liniile de comunicație dintre calculatoare și, ca urmare, la o viteză sporită de prelucrare.

Unul dintre neajunsurile aplicațiilor în rețea era (și mai este încă) viteza limitată de prelucrare a datelor, determinată de liniile de comunicație dintre calculatoarele rețelei. Oricât s-ar dezvolta calculatoarele unei rețele, viteza de prelucrare a datelor din rețea rămâne condiționată de legăturile între calculatoare.

Cablurile optice reprezintă una dintre tehnologiile care au condus la creșterea semnificativă a vitezei de transfer a datelor în rețea și, prin urmare, la creșterea vitezei de funcționare a aplicațiilor în rețea. O altă direcție de dezvoltare o reprezintă scăderea traficului în rețea, mai ales atunci când este vorba de baze de date de zeci și sute de megaocteți.

Conform modelului clasic de funcționare, pentru extragerea unor date dintr-o bază de date (care uneori poate fi foarte mare), este necesară copierea pe stația locală a tuturor datelor din baza de date și apoi prelucrarea lor local pentru extragerea rezultatelor dorite. Conform modelului client/server, stația locală transmite serverului o comandă prin care indică ce prelucrări trebuie efectuate și ce rezultate trebuie furnizate. Serverul preia comanda, o execută și transmite înapoi doar rezultatele. În acest fel, prin rețea circulă doar comanda și rezultatele prelucrării în locul bazei de date complete, traficul putând scădea de zeci, sute sau chiar mii de ori.

Prin urmare, modelul client/server implică existența unei aplicații client, care solicită unei aplicații server efectuarea unor operații. Aplicația server preia de la client comanda și o execută, transmițând înapoi rezultatele obținute. Aplicația server poate prelua comenzi de la mai multe aplicații client și răspunde de organizarea prelucrărilor, de prioritățile acordate fiecărui client în parte, de tratarea conflictelor date de accesul cvasisimultan al mai multor clienți la aceleași date.

Modelul client/server are și alte avantaje, pe lângă scăderea traficului prin rețea. Unul dintre acestea este scalabilitatea rețelei, adică posibilitatea creșterii performanțelor rețelei prin înlocuirea serverului cu unul mai performant. De această îmbunătățire vor beneficia toți utilizatorii, chiar dacă stațiile de lucru rămân aceleași. Să ne închipuim ce înseamnă aceasta pentru o rețea cu un server la 50 de stații de lucru: în loc să îmbunătățim 50 de calculatoare, îmbunătățim unul singur – costul este, evident, mult mai mic.

Un alt avantaj al aplicațiilor client/server este securitatea sporită. Aplicația server este una specializată, care controlează foarte eficient accesul la datele din rețea. Orice client are acces la date numai prin intermediul aplicației server (prin cereri adresate acesteia) și deci este obligat să respecte regulile impuse.

Modelul de funcționare client/server este specific rețelelor de calculatoare cu servere, dar el s-a extins și la nivelul aplicațiilor (pentru schimbul de date între aplicațiile care funcționează în același sistem de calcul). Un exemplu în acest sens este tehnologia DDE, prezentată într-un paragraf anterior. Această tehnologie folosește pentru comunicarea între aplicații modelul client/server.

În Visual FoxPro, modelul client/server este implementat prin intermediul limbajului SQL de prelucrare a bazelor de date relaţionale, pe baza căruia s-au construit vederile la distanţă. Protocolul folosit pentru implementarea modelului client/server este ODBC, protocol standard de comunicare cu serverele de baze de date.

Construirea vederilor la distanţă

Ce este o vedere la distanţă şi ce este o conexiune?

Una dintre metodele prin care este implementat modelul client/server în Visual FoxPro este reprezentată de vederile la distanţă.

Def

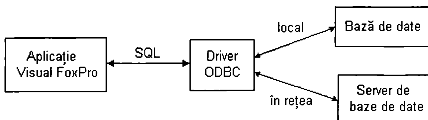
Vederile la distanţă *reprezintă un tip special de vederi, care sunt construite pe baza unor tabele de alte tipuri decât Visual FoxPro (create cu alt SGBD), aflate fie pe sistemul local, fie pe serverele reţelei din care face parte sistemul de calcul.*

În privinţa utilizării, ambele tipuri de vederi, simple şi la distanţă, sunt la fel, adică ele apar ca orice tabelă din care se pot citi şi în care se pot scrie date (folosind limbajul SQL). Diferenţa constă în ceea ce se află în spatele vederii, adică în modul în care se realizează comunicaţia dintre interfaţa sub formă de tabelă şi motorul care controlează accesul la date (sursa de date).

În cazul tabelelor simple, accesul la date se realizează direct, prin funcţiile motorului de date al sistemului Visual FoxPro. În cazul vederilor la distanţă, Visual FoxPro nu mai răspunde de citirea datelor de pe suportul de memorare, ci doar lansează către serverul de baze de date cereri cu privire la prelucrările dorite. Serverul (ca aplicaţie independentă) preia cererea, realizează prelucrarea corespunzătoare şi furnizează programului Visual FoxPro rezultatele. În acest caz, este deci necesar un mecanism (protocol) prin care să se controleze comunicaţiile între aplicaţii.

Pentru a putea utiliza tabele de alte tipuri decât cele din Visual FoxPro sistemul foloseşte un protocol special destinat comunicării cu serverele de baze de date, numit ODBC (**Open DataBase Connectivity** – conectivitate deschisă pentru baze de date). O dată cu instalarea SGBD se pot instala şi o serie de drivere ODBC, care să permită conectarea la diverse tipuri de baze de date. De asemenea, kitul de instalare al unui sistem informatic (generat cu Vrăjitorul respectiv) poate instala în sistemul de calcul destinaţie driverele ODBC folosite în aplicaţie.

Driverul ODBC reprezintă aplicaţia care rulează în sistemul de calcul local şi răspunde de comunicaţia cu serverul de baze de date. Din Visual FoxPro, driverul ODBC se vede ca o sursă de date de un anumit tip (în funcţie de tipul bazei de date).



Conectarea la un server de baze de date presupune specificarea de către utilizator a unor parametri, cum ar fi identificatorul utilizatorului, parola etc. Pentru a nu apărea obligaţia ca parametrii menţionaţi să fie specificaţi de fiecare dată când aplicaţia solicită acces la date de pe server, se utilizează o metodă automată de indicare a acestor parametri, conexiunile.

Def

***Conexiunile** reprezintă metoda prin care se pot memora parametrii de conectare a sistemului Visual FoxPro la o sursă de date ODBC instalată în sistemul de calcul, în vederea folosirii lor automate.*

O conexiune este memorată în fişierul bazei de date. În timpul rulării, conexiunea reprezintă conducta prin care circulă datele rezultate din comunicarea dintre vedere (Visual FoxPro) şi sursa de date (driverul ODBC).

Se poate construi o vedere la distanţă şi fără o conexiune, adică direct spre sursa de date (driverul ODBC), dar, în acest caz, la fiecare deschidere a vederii, trebuie precizaţi de către utilizator parametrii de conectare (numele utilizatorului, parola etc.).

În concluzie, pentru a putea construi în Visual FoxPro un mecanism de tip client/server, sunt necesare:

- instalarea şi configurarea driverelor ODBC pentru fiecare tip de bază de date folosită sau pentru fiecare tip de server folosit;
- stabilirea surselor de date pe baza driverelor ODBC disponibile;
- construirea vederilor la distanţă, fie prin intermediul conexiunilor, fie direct spre sursele de date.

Deşi în Visual FoxPro vederea la distanţă se foloseşte ca orice altă tabelă, ea se bazează şi pe componentele enumerate mai sus, care trebuie construite şi configurate de proiectantul aplicaţiei.

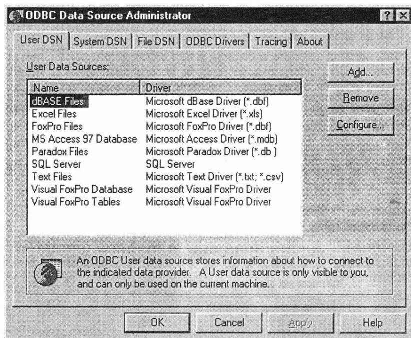
Configurarea sursei de date şi a driverului ODBC

Una dintre primele operaţii care trebuie realizate pentru construirea unei vederi la distanţă este configurarea surselor de date şi a driverelor ODBC, ţinând cont de faptul că modelul client/server din Visual FoxPro se bazează pe protocolul ODBC. Operaţia de configurare este externă mediului Visual FoxPro şi ţine de sistemul de operare (Windows).

La instalarea SGBD în sistemul de calcul se poate instala şi interfaţa ODBC, adică driverele ODBC de comunicare cu diferite servere de baze de date. O dată instalarea terminată, în aplicaţia Control Panel din Windows (folosită la configurarea sistemului) apare o pictogramă corespunzătoare aplicaţiei de configurare a interfeţei ODBC:

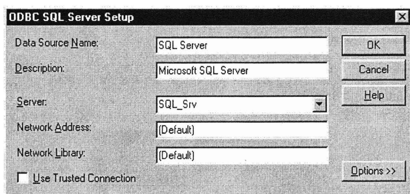


Acţionarea cu mouse-ul pe această pictogramă (prin clic dublu) determină deschiderea ferestrei de dialog pentru configurarea driverelor ODBC şi a surselor de date disponibile în aplicaţii.

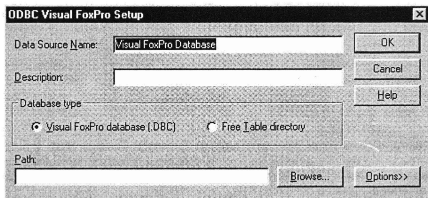


Fereastra conţine mai multe pagini alternative. În pagina **ODBC drivers** sunt prezentate driverele ODBC instalate în sistem. Dacă doriţi să folosiţi un anumit tip de bază de date, driverul corespunzător trebuie să se afle în această listă.

Fereastra **User DSN** (numele sursei de date a utilizatorului) este folosită pentru configurarea surselor de date. Fiecare sursă de date are la bază un driver ODBC. Butonul **Add** (adăugare) se foloseşte pentru adăugarea unei noi surse de date, iar butonul **Remove** (înlăturare) pentru ştergerea unei surse de date. Configurarea unei surse de date se face prin selectarea sursei din listă, urmată de acţionarea butonului **Configure** (configurare). Fereastra de dialog depinde de tipul sursei de date – de exemplu, cea de mai jos corespunde unui server SQL.



Observăm că în acest caz este necesară indicarea unor parametri specifici reţelei, precum numele serverului, adresa de reţea etc. Pentru bazele de date Visual FoxPro, fereastra de configurare a sursei de date arată ca în figura de mai jos:



Pe lângă alți parametri, precum numele sursei de date (**Data Source Name**) și descrierea acesteia (**Description**), este necesară și specificarea locului de unde vor fi preluate datele, adică unde se găsește baza de date de tip Visual FoxPro (în acest caz) din care vor fi preluate tabelele sursă în vederea la distanță.

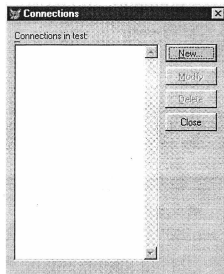
Construirea unei conexiuni

O dată parametrizate sursele de date folosite în aplicație (operație executată în afara SGBD Visual FoxPro), se poate trece la construirea vederilor la distanță și, dacă este necesar, a conexiunilor.

O conexiune este memorată într-o bază de date (în fișierul .DBC). Deci, crearea unei conexiuni este precedată de deschiderea bazei de date dorite, cu comanda:

```
MODIFY DATABASE <nume bază de date>
```

Urmează alegerea opțiunii **Connections** (conexiuni) a meniului **Database** pentru deschiderea ferestrei de dialog cu același nume, din interiorul căreia se controlează conexiunile bazei de date:



Pentru crearea unei noi conexiuni se folosește butonul **New** (nou), pentru modificarea uneia definite anterior se folosește butonul **Modify** (modificare), iar pentru ștergerea unei conexiuni se utilizează butonul **Delete** (ștergere). Conexiunea vizată este aleasă anterior din lista de conexiuni.

Specificarea parametrilor unei conexiuni (noi sau create anterior) se face în fereastra **Connection Designer** (proiectantul de conexiuni):

Există două metode de conectare la o sursă de date: prin furnizarea numelui utilizatorului, a parolei şi a bazei de date (butonul radio **Data source, userid, password** – sursa de date, identificator utilizator, parolă) sau prin trimiterea spre driver a unui şir de caractere în care sunt codificate datele de conectare (butonului radio **Connection string** – şir de conectare).

Parametrii respectivi sunt specificaţi în câmpurile de editare de sub butoanele radio. Sursa de date se alege din lista **Data source**, codul utilizatorului se introduce în câmpul de editare **Userid**, parola în câmpul **Password**, iar baza de date se precizează în câmpul **Database**. În cazul metodei şirului de conectare există un singur câmp de editare, în care se introduce şirul respectiv.

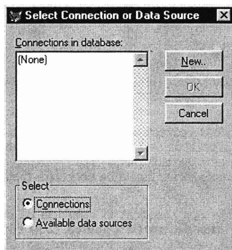
Există şi alţi parametri care trebuie specificaţi în celelalte obiecte de interfaţă ale ferestrei, dar nu vom intra aici în amănunte.

Construirea unei vederi la distanţă

Construirea unei vederi la distanţă este asemănătoare cu construirea vederilor simple (locale), cu excepţia părţii iniţiale în care se precizează parametrii de conectare la sursa de date.

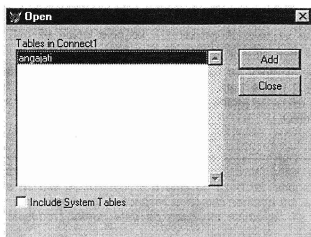
Construirea unei vederi la distanţă se realizează astfel:

- mai întâi se deschide baza de date în care urmează a fi memorată vederea respectivă;
- apoi se alege opţiunea **New Remote View** (o nouă vedere la distanţă) a meniului **Database**, după care pe ecran se deschide o fereastră de dialog în care se precizează dacă se va folosi Vrăjitorul (butonul **View Wizard**) sau Constructorul corespunzător (**New View**);
- vom prezenta metoda Constructorului, deoarece permite o configurare mai precisă, cu posibilităţi multiple de parametrizare. După acţionarea butonului **New View**, pe ecran apare următoarea fereastră de dialog, în care se indică modul în care se va realiza legătura dintre vedere şi sursa de date, direct sau printr-o conexiune:



- dacă se selectează butonul **Connections** (conexiuni), legătura se va realiza prin intermediul unei conexiuni din lista **Connections in database** (conexiuni în baza de date). Dacă încă nu s-a definit o conexiune în baza de date, se poate crea una prin acţionarea butonului **New**;

- dacă se selectează butonul **Available data sources** (surse de date disponibile), se va stabili o legătură directă între vedere şi sursa de date (driverul ODBC), fără a mai folosi ca intermediar conexiunea. După cum am mai spus, în acest caz este necesară precizarea de fiecare dată de către utilizator a parametrilor de conectare;
- după specificarea conexiunii, urmează indicarea tabelelor sursă din care vor fi preluate datele în noua vedere. Acestea se aleg dintre cele disponibile pentru conexiunea specificată în baza de date sursă. Fereastra de dialog din care se aleg tabelele respective arată ca în figura de mai jos:



- odată specificate tabelele sursă, se continuă cu celelalte operaţii specifice Constructorului de vederi (la fel ca în cazul vederilor simple); pentru detalii, se poate consulta paragraful dedicat acestui subiect.

Câteva aspecte legate de proiectarea aplicaţiilor client/server

În cele ce urmează, vom prezenta câteva aspecte de care trebuie să se ţină cont la proiectarea unei aplicaţii client/server, operaţie deloc uşoară, în care experienţa proiectantului este esenţială. Aplicaţiile client/server sunt mai dificil de realizat şi de depanat, din cauză că trebuie controlat nu numai modul de execuţie a unei aplicaţii locale (clientul), ci şi comunicarea sa cu aplicaţia server, care lucrează independent şi, poate, la distanţă.

În cadrul unei aplicaţii client/server există două tipuri de operaţii asupra datelor din bazele de date, cele executate local de către aplicaţie şi cele executate la distanţă de către server. Unul dintre secretele eficienţei unei aplicaţii client/server este modul în care se împart sarcinile de prelucrare între cele două părţi implicate, clientul şi serverul.

O altă problemă care trebuie rezolvată de proiectant este localizarea datelor: fie local, pe suportul de memorare al staţiei de lucru, fie pe server. În acest sens, trebuie stabiliţi anterior o serie de parametri, printre care: cine are acces la datele respective, care este gradul de modificare a datelor, care sunt prelucrările ce trebuie executate asupra lor etc.

O schimbare majoră în proiectarea aplicaţiilor client/server faţă de cele clasice este trecerea de la prelucrări punctuale, asupra unor înregistrări individuale, la prelucrări pe blocuri de înregistrări. Cu alte cuvinte, prelucrările de pe server se vor realiza prin comenzi SQL (care se execută pe server), iar cele locale se pot realiza atât prin comenzi SQL, cât şi prin comenzi clasice de tipul GOTO, REPLACE etc.

În privinţa prelucrărilor pe server, principalul obiectiv este acela de a aduce pe staţia locală cât mai puţine date, numai cele care sunt necesare. În acest fel este diminuat traficul în reţea, se profită din plin de puterea serverului, iar viteza de rulare a aplicaţiei creşte simţitor.

Exemplu

De exemplu, dacă într-o formă se realizează selectarea unor produse dintr-o anumită categorie, iar baza de date a produselor se află pe server, este de preferat a se citi anterior categoria respectivă, pentru ca apoi, în lista formei (deci local) să fie aduse din baza de date numai produsele din categoria precizată.

Varianta nefericită ar fi aceea de a transfera toate produsele din baza de date şi de a realiza filtrarea tabelei local.